AFIPS

CONFERENCE PROCEEDINGS

VOLUME 54

1985

NATIONAL COMPUTER CONFERENCE

AFIPS
CONFERENCE PROCEEDINGS

1985
NATIONAL COMPUTER CONFERENCE

July 15–18, 1985
Chicago, Illinois

ANTHONY S. WOJCIK
Editor and Program Chairman

KARL E. MARTERSTECK
Conference Chairman

Sponsored by
AMERICAN FEDERATION OF INFORMATION
PROCESSING SOCIETIES, INC.
ASSOCIATION FOR COMPUTING MACHINERY
DATA PROCESSING MANAGEMENT ASSOCIATION
IEEE COMPUTER SOCIETY
SOCIETY FOR COMPUTER SIMULATION

AFIPS PRESS
1899 PRESTON WHITE DRIVE
RESTON, VIRGINIA 22091

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1985 National Computer Conference or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 80-649583
ISSN 0095-6880
ISBN 0-88283-046-5

AFIPS PRESS
1899 Preston White Drive
Reston, Virginia 22091

The National Computer Conference
sponsored by

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC.
ASSOCIATION FOR COMPUTING MACHINERY
DATA PROCESSING MANAGEMENT ASSOCIATION
IEEE COMPUTER SOCIETY
SOCIETY FOR COMPUTER SIMULATION

Printed in the United States of America

Preface

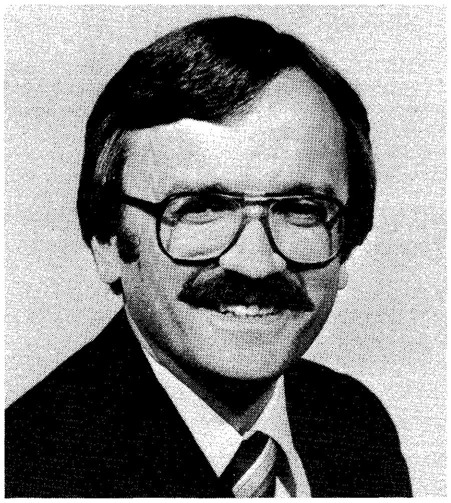
KARL E. MARTERSTECK
1985 NCC Chairman

The *Proceedings* of the 1985 National Computer Conference presents the most comprehensive view of current developments in the computing industry to date. This year's technical program has captured the breadth and depth of the ever-increasing variety of applications of computing technology in our world—hence our 1985 theme "Technology's Expanding Horizons"!

Needless to say, many diverse elements must be smoothly brought together to have a successful conference, particularly one as significant as the NCC. There are technical programs, Professional Development Seminars, technical exhibits, and a variety of special events such as Pioneer Day, all functioning in a spacious physical facility in an exciting host city. But most of all, the element that uniquely makes the conference something of value and something special is the *people* involved in making it happen!

Our NCC'85 team was a particularly professional and dedicated group. The members of the Conference Steering Committee and the various committees they chaired created and implemented a detailed plan to insure that the conference would satisfy the computing information needs of all attendees. The AFIPS staff provided magnificent support in working with the Conference Steering Committee. The NCC Committee and the NCC Board gave valuable guidance to all of us. Finally, and most significantly, there are you, the attendees. Without your enthusiastic participation in the technical sessions as well as your energetic interaction with the exhibitors, the conference would fail in its most fundamental objective—the exchange of ideas and information regarding the issues and challenges facing the computing community today.

Hopefully, the *Proceedings* will provide you with a lasting, valuable reference to the key ideas and information you acquired during NCC'85.



Introduction

ANTHONY S. WOJCIK
1985 NCC Program Chairman

The technical program of an NCC provides the attendee the opportunity to discover the latest advances and trends over the broad spectrum of what can be called the information processing industry. Among conferences, the NCC is unique because of this wide scope. As a result, the task of the NCC Program Committee is to divide the industry into areas, or tracks, and to build a program along the selected tracks that will be representative of the industry. NCC attendees have diverse backgrounds, and the program must appeal to this diversity.

The program for NCC '85 consists of 84 sessions, divided among nine tracks, with several special sessions of particular interest. The tracks reflect areas that traditionally have been important to NCC attendees. This year the Conference theme, "Technology's Expanding Horizons," has influenced the development of each track. The impact of technological expansion has resulted in more processing capability through the development of denser circuitry, richer software tools and packages, more effective systems for linking computers, and innovative systems incorporating novel architectures. The Program Committee has structured many sessions to reflect these ideas.

Each year far more papers are submitted than can possibly be accepted for the NCC. With the assistance of hundreds of referees, whose participation is absolutely essential, the Program Committee has selected the papers that appear in this *Proceedings*. These papers reflect the broad scope of the NCC and emphasize the today and the tomorrow of our industry.

This *Proceedings* also includes a special section focusing on the Conference's Pioneer Day theme, the First Generation in Illinois. In addition to three papers written by scientists who were part of this early development, there are reprints of several classic papers that provide further insight into the development of several early computers. The Program Committee hopes that this special section will give us a deeper appreciation of our industry's past.

It has been a challenge to coordinate the development of the program for this year's Conference. I extend my sincerest appreciation and thanks to each of the members of the Program Committee. Without their assistance and counsel, the challenge of developing the program would not have been met.

CONTENTS

PIONEER DAY: THE FIRST GENERATION IN ILLINOIS

Reprints of classic papers

ARTIFICIAL INTELLIGENCE

FUTURE ARCHITECTURES AND SUPERCOMPUTERS

EDUCATIONAL AND SOCIETAL ISSUES

END USER COMPUTING

INFORMATION SYSTEMS MANAGEMENT

BUSINESS APPLICATIONS

PERSONAL COMPUTING

SOFTWARE SYSTEMS

NETWORKING

PIONEER DAY: THE FIRST GENERATION IN ILLINOIS

MARGARET K. BUTLER, Track Chair
Argonne National Laboratory
Argonne, Illinois

A decade of ORACLE experience

by EARL W. BURDETTE
U.S. Department of Energy
Oak Ridge, Tennessee
and
RUDOLPH J. KLEIN
Monarch Marking Systems
Dayton, Ohio

ABSTRACT

This paper is a chronological story of the design, construction, and operation of the ORACLE over its ten year life cycle. The authors participated in all three stages of ORACLE's life cycle, and the material presented was extracted from personal files of the authors and annual reports of the Oak Ridge National Laboratory.

This overview and description of the ORACLE reveals the nature of a first generation computer and identifies the major differences in engineering design and construction of the ORACLE over that of the other IAS (Institute for Advanced Study)-type computers.

The important lessons learned and legacies listed should be of value to future generations from a historical viewpoint, and for those concerned with how the computer explosion of the past thirty-five years was effected by its origins.

In conclusion, the authors make some philosophical remarks called *pioneer reflections on our contemporary society.*

INCEPTION OF THE OAK RIDGE AUTOMATIC COMPUTER AND LOGICAL ENGINE (ORACLE), 1949

In May, 1949, a mathematics and computing panel was organized at ORNL (Oak Ridge National Laboratory) to provide services to all research divisions of the laboratory. Dr. Alston S. Householder was named Chief of the panel which consisted of mathematicians and computer scientists with Professor John von Neumann as Consultant. The purpose of the panel was to perform computations and provide a consulting service on mathematical, statistical, and computational problems arising in the research divisions, and to engage in mathematical research on problems relevant to the overall research program of the laboratory. A large class of such problems centered on the effective use and evaluation of the Monte Carlo method, and numerical methods in general.

While ORNL had no high-speed computing machinery at that time, various facilities were available for its use; there was IBM equipment at both the K-25 and Y-12 plants; the Mark I at Harvard was used for computational research. The NEPA (Nuclear Energy Propulsion Agency), and AEC (Atomic Energy Commission) requested that ORNL pursue a plan to acquire a high-speed computer for their common use and arrange for the use of other existing machines until one became available in Oak Ridge.

During early 1949, P. R. Bell of the Physics Division, and A. S. Householder and Lewis Nelson of the Math Panel were asked to examine the current development of high speed computing machinery. They subsequently visited the RCA Laboratory in Princeton, NJ, the Computing Machinery Laboratory of the Institute for Advanced Study in Princeton, and the Eckert-Mauchly Computer Machinery Corporation in Philadelphia, PA. Consideration was also given to a Raytheon computer, and a Sylvania Electric Harvard Mark III computer.

The Raytheon computer was first recommended for purchase by the ORNL committee; however, because excellent progress was being made in developing the Williams Tube electrostatic fast memory, the committee reversed its decision and recommended that ORNL make an agreement with ANL (Argonne National Laboratory) for the construction of an IAS electronic digital computer for Oak Ridge.

ANL had organized an experienced electronic computer group under Dr. Jeffrey C. Chu, and planned to continue permanent research on the development of electronic computers.

A formal agreement was signed between ORNL and ANL for the design and construction of an Oak Ridge computer based on work in progress on the ANL AVIDAC and the IAS computer at Princeton. As part of that agreement, ORNL was to provide engineers from Oak Ridge to assist ANL engineers in the design and construction of the Oak Ridge computer. Four ORNL engineers, Earl W. Burdette, William J. Gerhard, Rudolph J. Klein, and James W. Woody took up temporary residence in Chicago to work at the Argonne National Laboratory, assisting in the completion of the AVIDAC and creating the ORACLE.

DESIGN AND CONSTRUCTION AT ARGONNE, 1950–1953

The ORACLE was designed and constructed at Argonne by a staff of engineers from Argonne and Oak Ridge National Laboratories under the direction of J. C. Chu. In logic design, the ORACLE was a modification of the AVIDAC, which in turn was a modification of the computer at the Institute for Advanced Study at Princeton. However, the engineering design of the ORACLE was changed greatly for the purpose of improving reliability, ease of maintenance, and speed of operation. These goals were accomplished according to the records kept during the seven years of ORACLE operation at Oak Ridge.

The ORACLE, which was built at a cost of $350,000, contained three key features which gave it advantage over other computing devices of its day. First, its internal memory system had the greatest capacity of any previous high-speed general-purpose computer; at 2048 10-digit decimal numbers, it had twice the capacity of other computers of this type and about eight times that of most of the earlier machines. Second, the ORACLE was provided with an internally-controlled 42-track magnetic-tape auxiliary memory system which allowed for the intermediate storage of four million words of calculations and data; this was the largest on-line storage system operational on a computer at that time. Third, the ORACLE was the fastest of the general-purpose computers. It could multiply 9-digit numbers such as 999,999,999 by 999,999,999 in less than 1/2000 of a second. The addition of two 10-digit decimal numbers took place in 5/1,000,000 of a second.

The computer basically consisted of seven major sections (which still holds for most computers today) as shown in Figure 1.

1. An arithmetic unit which performs addition, subtraction, multiplication, division, and other arithmetic and logical operations;
2. A control section which determined the sequence in which the program of instructions would be carried out;

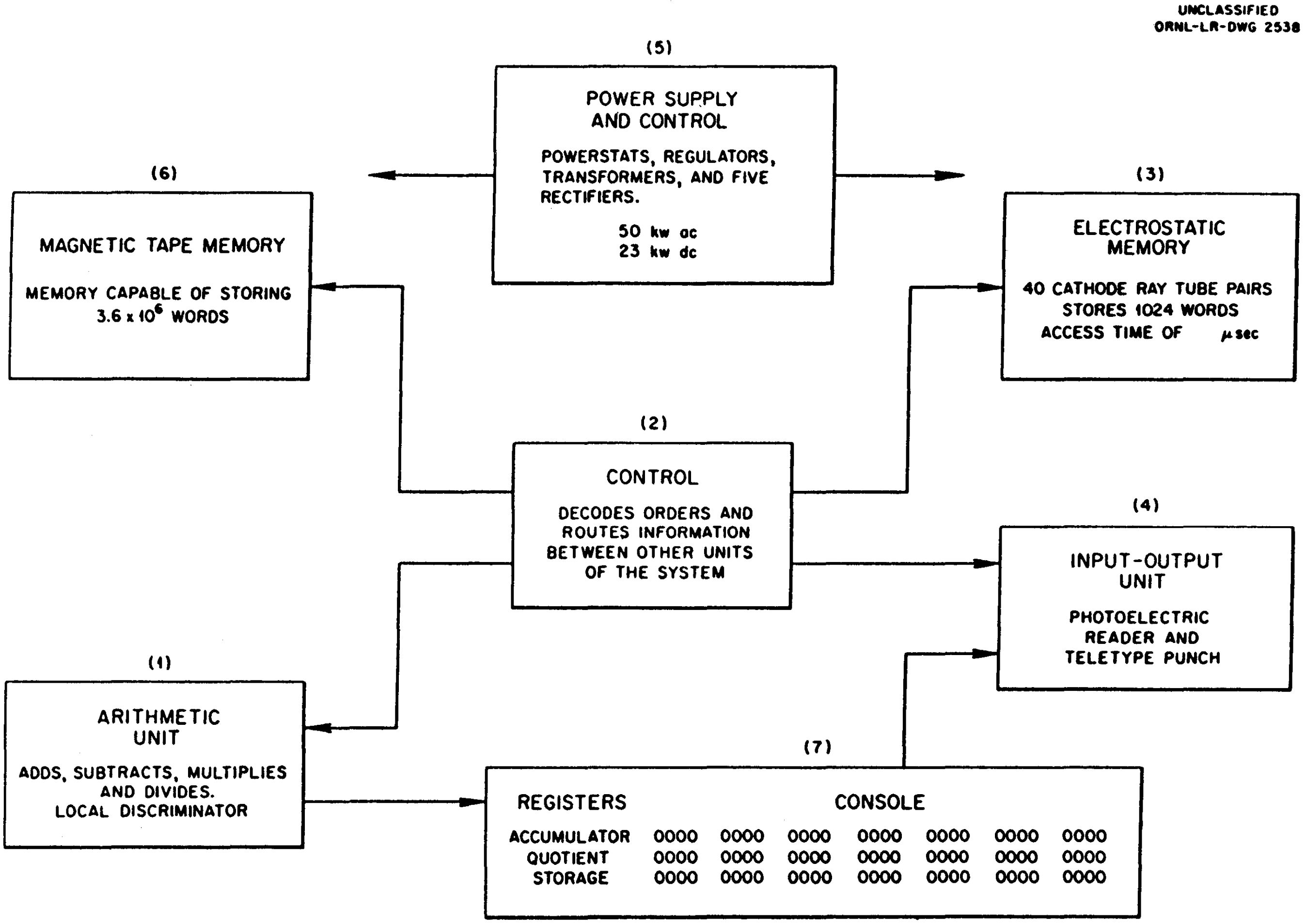

UNCLASSIFIED
ORNL-LR-DWG 2538
(5)
POWER SUPPLY AND CONTROL
POWERSTATS, REGULATORS, TRANSFORMERS, AND FIVE RECTIFIERS.
50 kw ac
23 kw dc
(6)
MAGNETIC TAPE MEMORY
MEMORY CAPABLE OF STORING 3.6 x 10⁶ WORDS
(3)
ELECTROSTATIC MEMORY
40 CATHODE RAY TUBE PAIRS
STORES 1024 WORDS
ACCESS TIME OF μsec
(2)
CONTROL
DECODES ORDERS AND ROUTES INFORMATION BETWEEN OTHER UNITS OF THE SYSTEM
(4)
INPUT-OUTPUT UNIT
PHOTOELECTRIC READER AND TELETYPE PUNCH
(1)
ARITHMETIC UNIT
ADDS, SUBTRACTS, MULTIPLIES AND DIVIDES.
LOCAL DISCRIMINATOR
(7)
REGISTERS
CONSOLE
ACCUMULATOR 0000 0000 0000 0000 0000 0000 0000
QUOTIENT 0000 0000 0000 0000 0000 0000 0000
STORAGE 0000 0000 0000 0000 0000 0000 0000

Figure 1—Block diagram of ORACLE operation

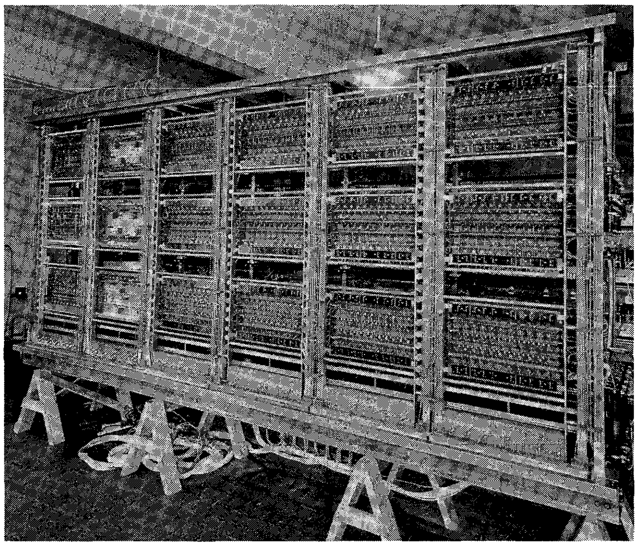


Figure 2—Arithmetic unit

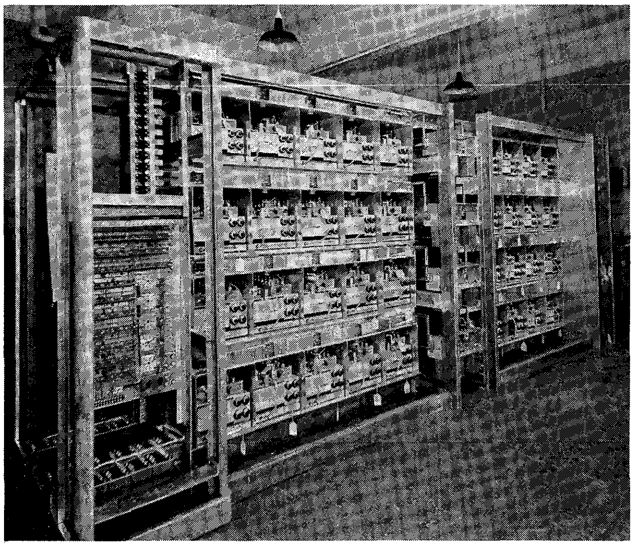


Figure 3—Memory unit

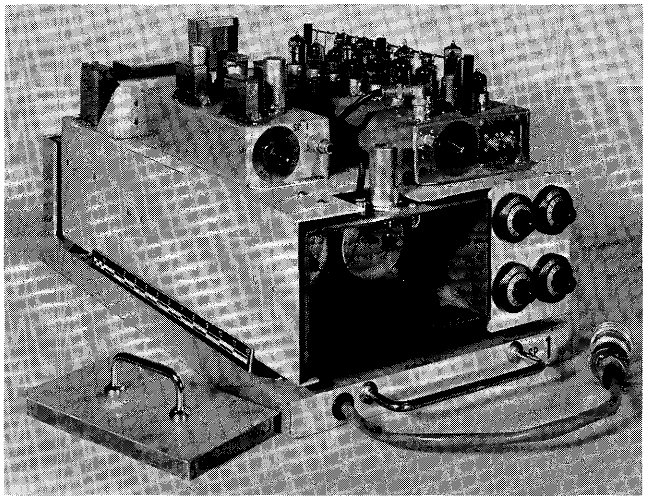


Figure 4—ORACLE memory individual "bit drawer"

3. The internal memory; a fast parallel-access electrostatic internal memory to store instructions and data;
4. An input-output section where information is staged and interfaced into or out of the outside world;
5. A power supply;
6. A large-capacity auxiliary magnetic-tape memory to hold intermediate calculations and data; and
7. An operating console to provide a convenient means to manually control the computer.

Figure 5—Margaret Butler and Bud Klein servicing a troublesome memory unit

Figures 2 through 8 are pictures of the major units of the ORACLE designed and built at Argonne. All of these units were integrated into a system that allowed good access for maintenance, adequate cooling, efficient power, orderly signal distribution, and ease of operation.

The Arithmetic Unit shown in Figure 2 consisted of three banks of 40 asymmetrical flip-flops, the "A" and "Q" register connected by 40 parallel logical adders, an "M" register to hold instructions and operands, and four driver-chassis. A similar general architecture is still used today in microprocessor chips.

The electrostatic storage memory unit shown in Figures 3, 4 and 5 was a fast parallel-access internal memory consisting of 43 plug-in memory stages (40 in service and three in warm-up), the beam deflection system, and the memory monitor. This Williams-type storage system used two three-inch cathode-ray tubes per stage, and was organized into 2048 40-bit computer words, with parallel access to any word within the 20-microsecond memory cycle time.

The auxiliary memory (magnetic tape) shown in Figures 6, 7, and 8 had four drive units with removable 1,000 foot reels of two-inch wide tape per unit. Each tape had 42 channels with a packing density of 100 words per inch. Total storage capacity was 3,600,000 words or 18,000,000 bytes.

Automatic control of computer operations was accomplished by four control sections, fast memory (electrostatic), arithmetic, auxiliary memory (magnetic tape) and input-

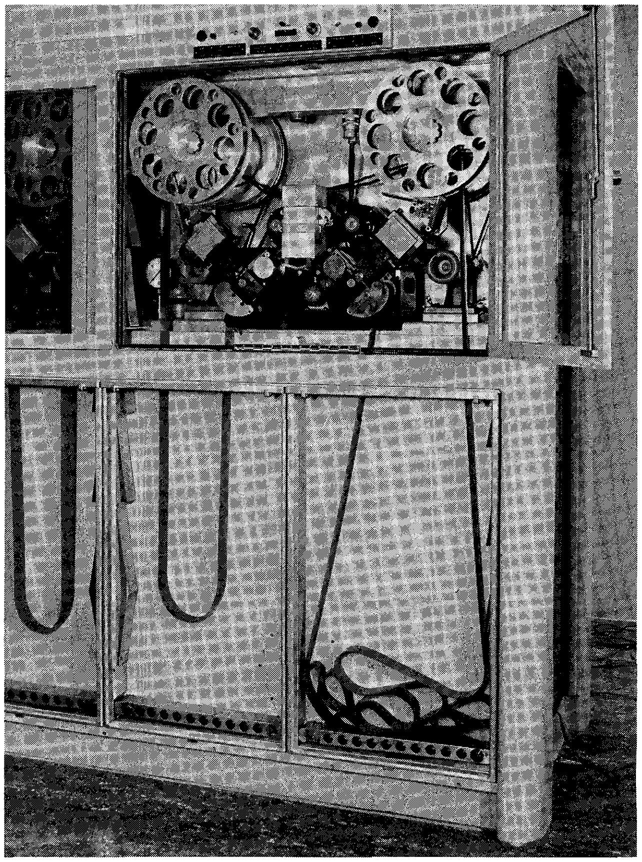


Figure 6—Auxiliary magnetic tape information storage unit

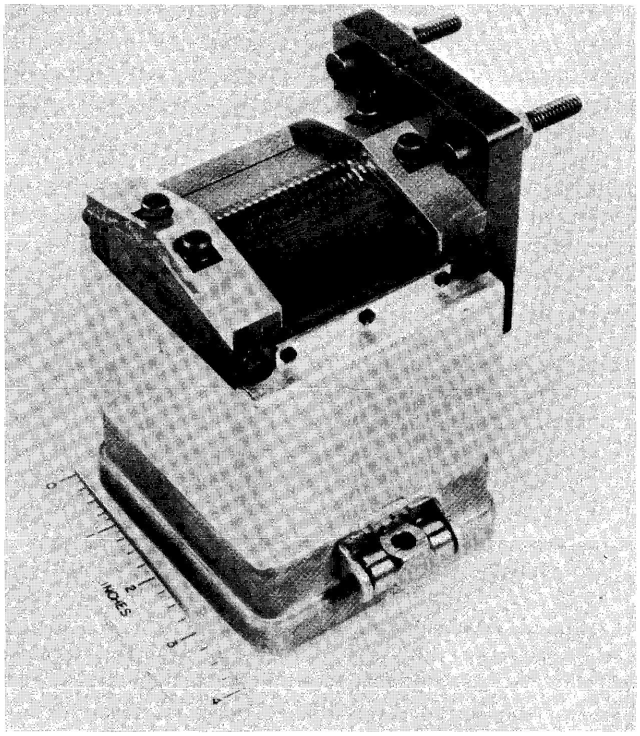


Figure 7—42-track magnetic tape recording head

output. These control sections were all synchronized to allow a minimum of dead-time operation of each unit. The arithmetic, auxiliary memory, and input-output units had asynchronous controls that permitted each unit to proceed independently. The fast memory was synchronous, to provide continuous regeneration of information stored on the CRTs when the other units were not using the memory to read or write information.

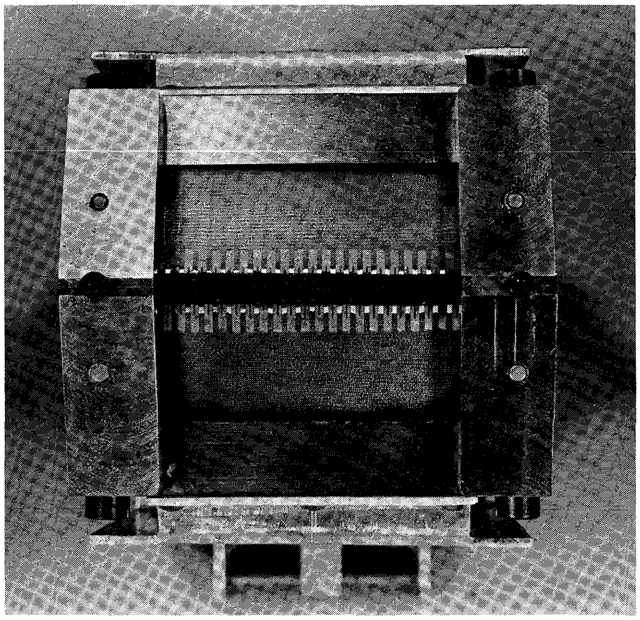


Figure 8—42-track magnetic tape recording head

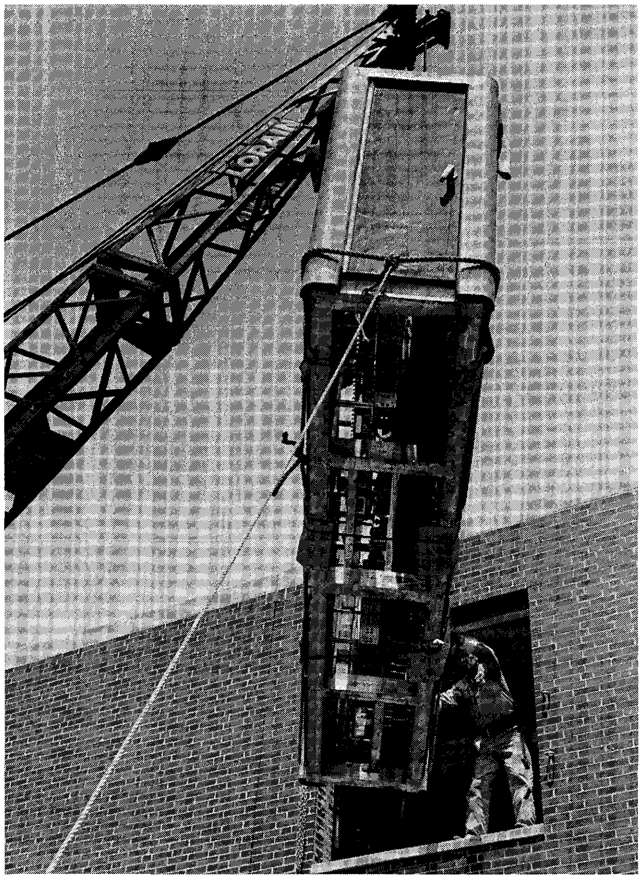


Figure 9—Moving ORACLE into ORNL, October, 1953

The arithmetic unit circuitry, control circuitry, electrostatic memory, magnetic tape auxiliary memory, and input-output units were all to some extent different in engineering design and construction from other Institute of Advanced Study computers.

The arithmetic unit used the three dimensional chassis layout of most of the IAS computers; however, the circuits and

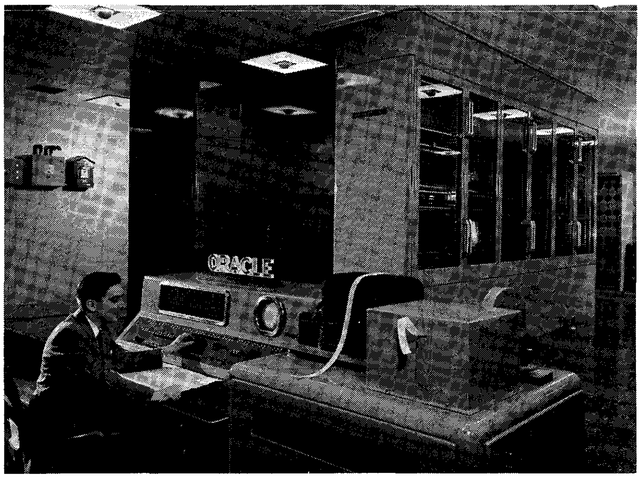


Figure 10—ORACLE, 1954

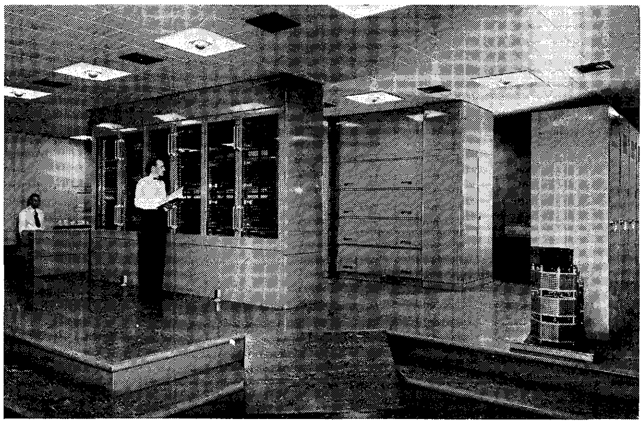


Figure 11—ORACLE raised floor installation, 1954

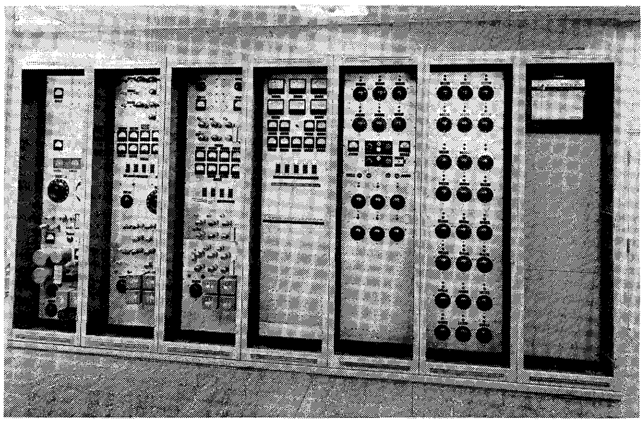


Figure 12—ORACLE power control system

components contained DC and AC coupling circuits, synchronous and asynchronous circuits, logical and analog gating circuits, semi-conductor diodes, vacuum tubes, and asymmetrical rather than symmetrical flip-flops.

The ORACLE became operational at Argonne during the summer of 1953, and was used for testing and subroutine writing until it passed its final tests in early fall of that year.

INSTALLATION AT OAK RIDGE, 1953–1954

After two years of design and construction at the Argonne National Laboratory, the ORACLE arrived at Oak Ridge

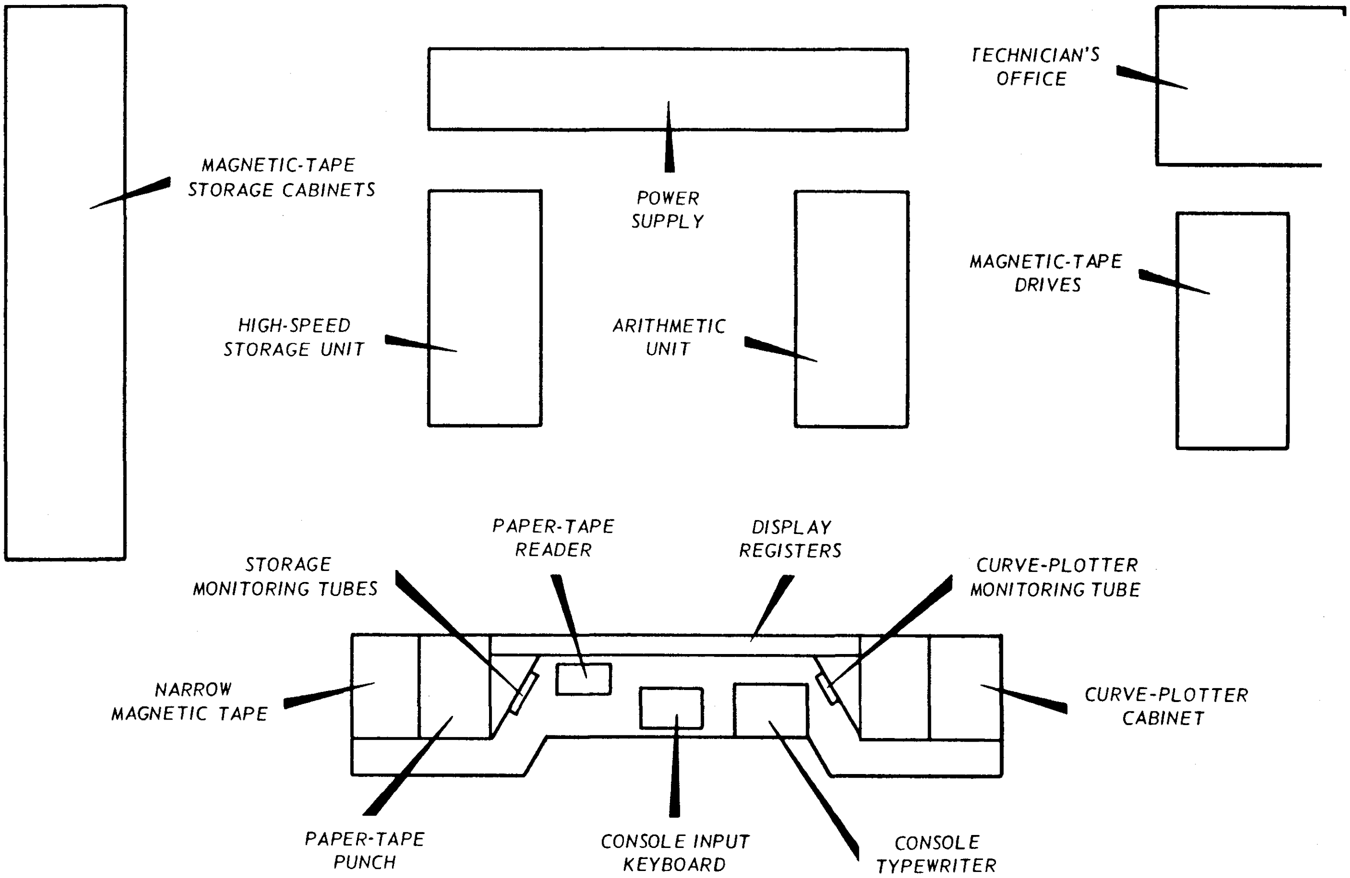


Figure 13—Floor plan of the computer room, showing the primary ORACLE components (paper tape preparation and printing machines are in an adjoining room)

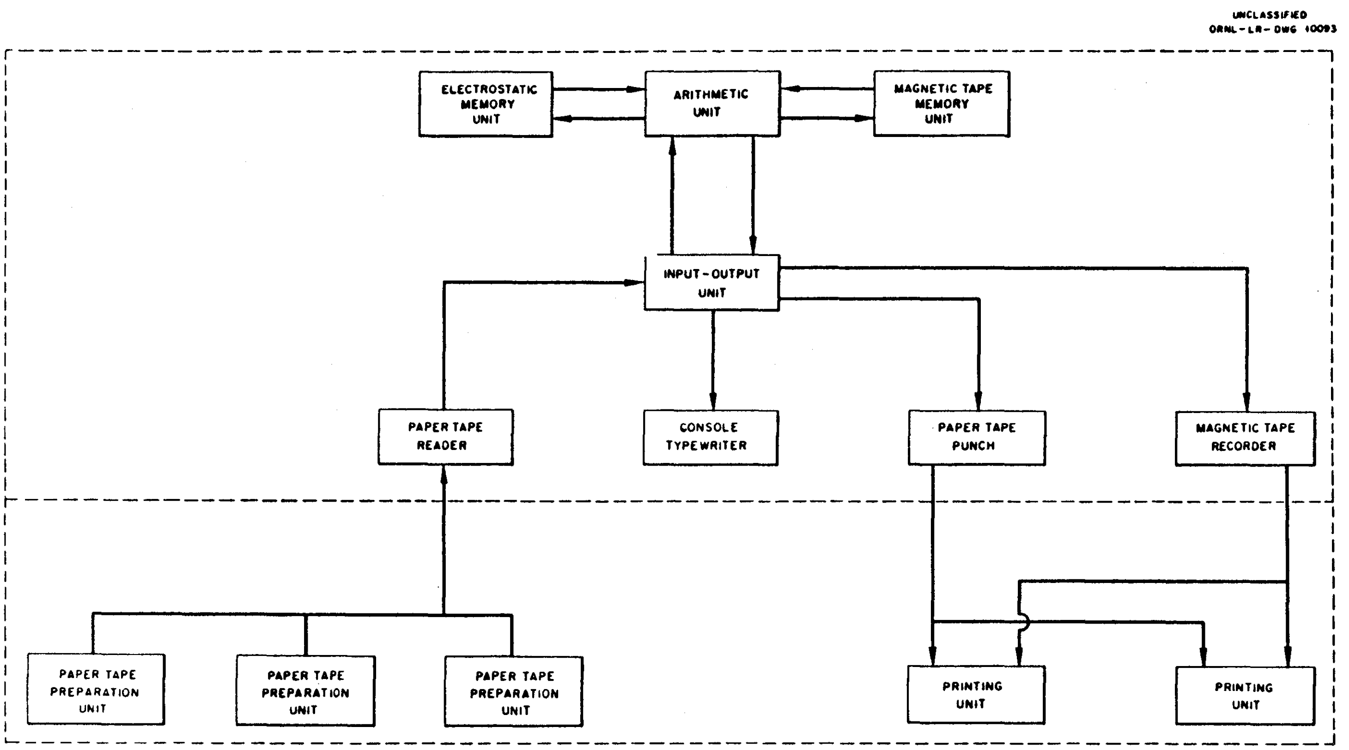


Figure 14—ORACLE auxiliary equipment block diagram

National Laboratory on October 8, 1953 and began full scale operation on February 4, 1954 after four months of installation, test, and checkout.

The physical size, complexity, and delicate construction of the computer made installation in the laboratory building a major undertaking (see Figure 9). Exceptional care was exercised in the installation to provide protection and a controlled environment for the machine during operation.

The initial computer installation included the electrostatic memory unit, the arithmetic unit, power supply cabinets and operating console as shown in Figure 10. The auxiliary magnetic tape unit was installed later as were a CRT plotter and several major enhancements of the input-output system.

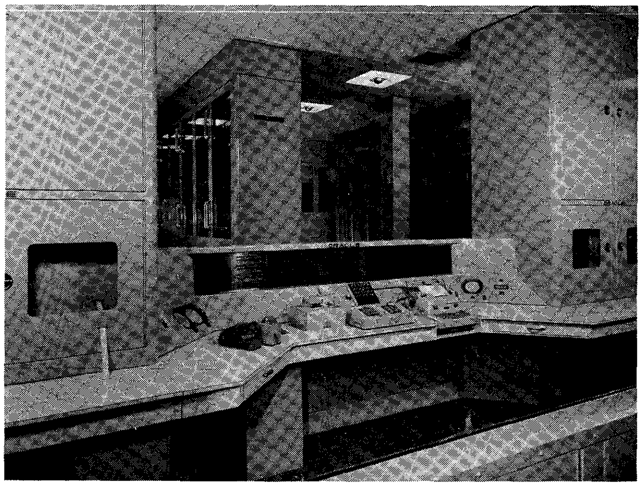


Figure 15—ORACLE enhancements

Figure 11 shows the initial installation with the computing units installed on a raised floor which contained the cabling between units and the air conditioning ducts to the computer units. A 40-ton air-conditioning system provided a closed air conditioned computer room. A 75-kW power system, located in the power cabinets at the back of the computer room supplied the necessary regulated and unregulated power sources for the computer units (see Figure 12).

ENHANCEMENTS AT ORNL, 1954–1956

During the design and construction work at Argonne, the primary emphasis was on the heart of the computer, the arithmetic unit, electrostatic memory, and auxiliary magnetic units. The slow teletype paper tape input-output system was only the bare minimum necessary to get computer programs and data into the computer and some of the data out of it.

The initial system installed at Oak Ridge had a teletype paper tape reader and punch which operated at very slow speeds. After the computer was installed, checked out, and turned over to operations, the Oak Ridge engineering group turned their efforts toward designing and building a modern input-output system.

The objectives of the new input-output system were (1) to improve flexibility, by allowing alphabetic characters and special symbols to be used in the computer in addition to the numeric characters already available, (2) to increase the output speed by a factor of 50, and (3) to increase the reliability

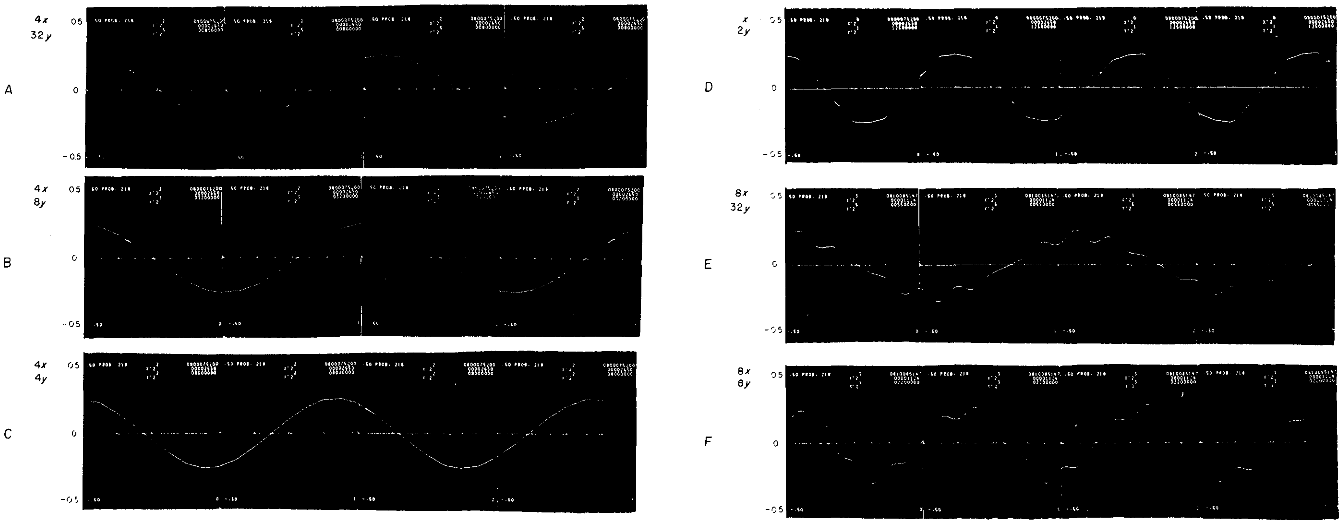
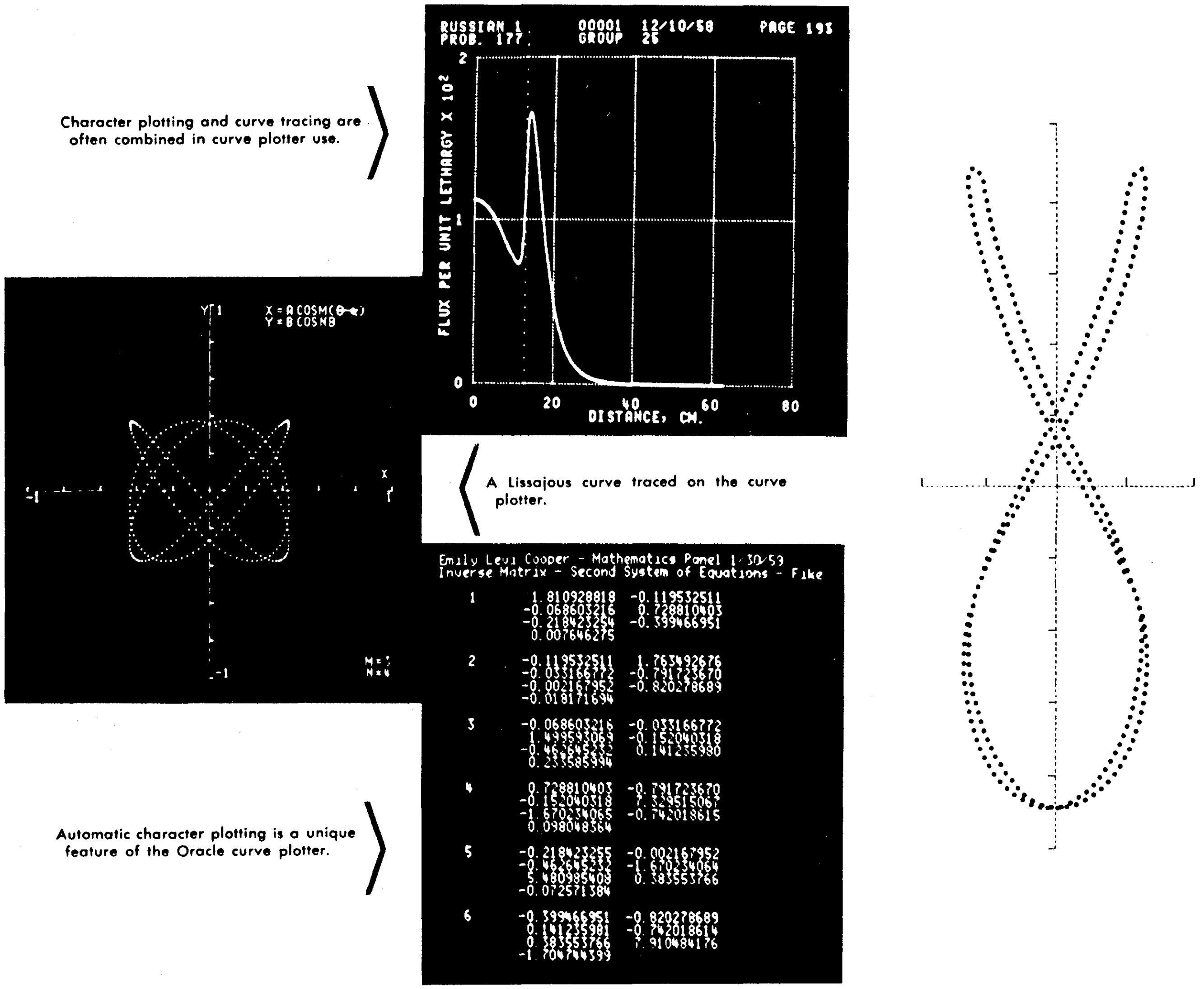


Figure 16—ORACLE CRT plotter output



Character plotting and curve tracing are often combined in curve plotter use.

A Lissajous curve traced on the curve plotter.

Automatic character plotting is a unique feature of the Oracle curve plotter.

```
Emily Levi Cooper - Mathematics Panel 1/30/59
Inverse Matrix - Second System of Equations - Fike

    1       1.810928818  -0.119532511
           -0.068603216   0.728810403
           -0.218423254  -0.399466951
            0.007646275

    2      -0.119532511   1.763492676
           -0.033166772  -0.791723670
           -0.002167952  -0.820278689
           -0.018171694

    3      -0.068603216  -0.033166772
            1.499593069  -0.152040318
           -0.462645232   0.141235980
            0.233585994

    4       0.728810403  -0.791723670
           -0.152040318   7.329515067
           -1.670234065  -0.742018615
            0.098048364

    5      -0.218423255  -0.002167952
           -0.462645232  -1.670234064
            5.480985408   0.383553766
           -0.072571384

    6      -0.399466951  -0.820278689
            0.141235981  -0.742018614
            0.383553766   7.910484176
           -1.704744399
```

Figure 17—ORACLE CRT output

of data transfer into and out of the computer by using a parity checking system.

Figure 13 is a floor plan of the enhanced computer system with the new input-output system. Figure 14 is a block diagram of the new input-output system. All the paper tape handling and copy equipment (typewriters and magnetic tape) were speeded up, and a very fast cathode-ray tube output device was developed. Figure 15 shows the finished integrated operating console and new input-output system equipment.

The cathode-ray tube output device was both a curve plotter and character plotting device. It included an electrostatically deflected and focused cathode-ray plotting tube, a digital-to-analog converter for allowing the computer to control the deflection of the electron beam in the plotting tube, an automatic camera for making permanent records of the plot, and a monitor tube for allowing constant visual inspection during the plotting operation.

The plotting speed was 200 microseconds per point, thus permitting a complete graph with axis and labeling to be plotted in less than one second. The curve-plotter camera had a film capacity of 200 exposures. Figures 16, 17, and 18 are examples of graphic and character output.

SUMMARY OF OPERATING EXPERIENCE, 1954–1961

Figure 19 is a bar chart of the use of the ORACLE from 1954 to 1961. Figure 20 shows a typical year of use by the research divisions of the Laboratory.

The ORACLE operated on a three-shift five-day per week schedule. Production, program debugging, preventive maintenance, and special project work was scheduled on a weekly basis. The average up-time for all planned work usually exceeded 95%. This excellent up-time ratio (especially for vacuum-tube machines) was primarily due to a rigorous preventive maintenance program.

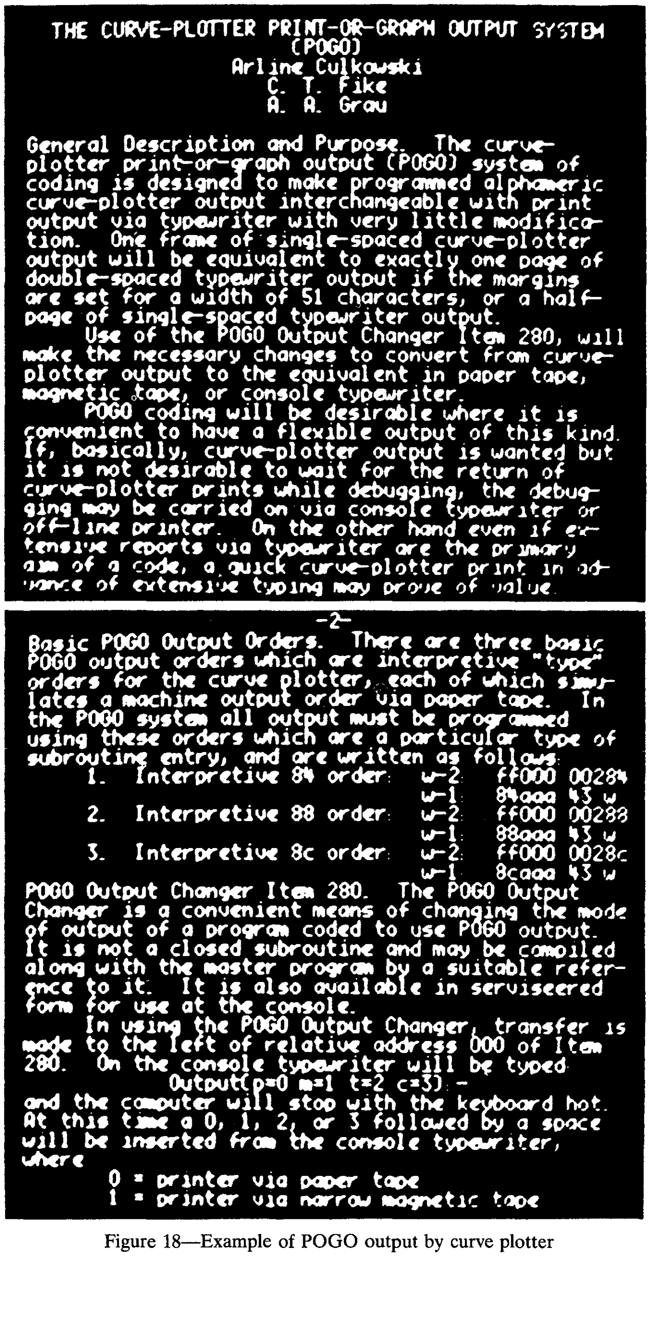
During the seven years of ORACLE operation, an efficient system of machine maintenance was developed. Each week, 15 hours were reserved to provide preventive maintenance. This maintenance included the following:

1. Tube replacement (approximately 150 per week) to ensure that no tube remained in the machine longer than 6000 hours;
2. Only tested tubes were installed after periods of shelf test and burn in;
3. Software diagnostic routines were developed to extensively test and determine the operating condition of the machine;
4. Magnetic heads and transports were cleaned three times each day;
5. Voltage and current readings were logged each week to detect degradation conditions; and
6. Daily and weekly records were kept on each unit to plot and analyze trends in condition of the machine so that preventive maintenance could be adjusted based on experience.

The ORACLE was retired from active service in the fall of 1962 after 8 years of work-horse duty as the prime scientific computing resource supporting the research efforts of an approximately 2,000-person laboratory staff. This remarkable endurance record is a testament to the vision and engineering competence of the combined Argonne and Oak Ridge design team, and to the aggressive updating and maintenance efforts of the Oak Ridge operational team. Many aspects of this pioneering computer were novel for its day, but it is difficult to detail the exact genealogy of thinking which has led to similar features in present day computer systems. Some idea of the position of the ORACLE in the fermentation of evolving electronic computers, beginning with ENIAC, may be seen in Figure 21.

LESSONS LEARNED AND LEGACIES LEFT

During the design, construction, and operation of the ORACLE, three primary lessons became very evident.



```
   THE CURVE-PLOTTER PRINT-OR-GRAPH OUTPUT SYSTEM
                       (POGO)
                  Arline Culkowski
                     C. T. Fike
                     A. A. Grau

  General Description and Purpose.  The curve-
  plotter print-or-graph output (POGO) system of
  coding is designed to make programmed alphameric
  curve-plotter output interchangeable with print
  output via typewriter with very little modifica-
  tion.  One frame of single-spaced curve-plotter
  output will be equivalent to exactly one page of
  double-spaced typewriter output if the margins
  are set for a width of 51 characters, or a half-
  page of single-spaced typewriter output.
       Use of the POGO Output Changer Item 280, will
  make the necessary changes to convert from curve-
  plotter output to the equivalent in paper tape,
  magnetic tape, or console typewriter.
       POGO coding will be desirable where it is
  convenient to have a flexible output of this kind.
  If, basically, curve-plotter output is wanted but
  it is not desirable to wait for the return of
  curve-plotter prints while debugging, the debug-
  ging may be carried on via console typewriter or
  off-line printer.  On the other hand even if ex-
  tensive reports via typewriter are the primary
  aim of a code, a quick curve-plotter print in ad-
  vance of extensive typing may prove of value.

                         -2-
  Basic POGO Output Orders.  There are three basic
  POGO output orders which are interpretive "type"
  orders for the curve plotter, each of which simu-
  lates a machine output order via paper tape.  In
  the POGO system all output must be programmed
  using these orders which are a particular type of
  subroutine entry, and are written as follows:
       1.  Interpretive 84 order:  w-2:  ff000 00284
                                   w-1:  84aaa 43 w
       2.  Interpretive 88 order:  w-2:  ff000 00288
                                   w-1:  88aaa 43 w
       3.  Interpretive 8c order:  w-2:  ff000 0028c
                                   w-1:  8caaa 43 w
  POGO Output Changer Item 280.  The POGO Output
  Changer is a convenient means of changing the mode
  of output of a program coded to use POGO output.
  It is not a closed subroutine and may be compiled
  along with the master program by a suitable refer-
  ence to it.  It is also available in serviseered
  form for use at the console.
       In using the POGO Output Changer, transfer is
  made to the left of relative address 000 of Item
  280.  On the console typewriter will be typed:
                 Output(p=0 m=1 t=2 c=3): -
  and the computer will stop with the keyboard hot.
  At this time a 0, 1, 2, or 3 followed by a space
  will be inserted from the console typewriter,
  where
        0 = printer via paper tape
        1 = printer via narrow magnetic tape
```

Figure 18—Example of POGO output by curve plotter

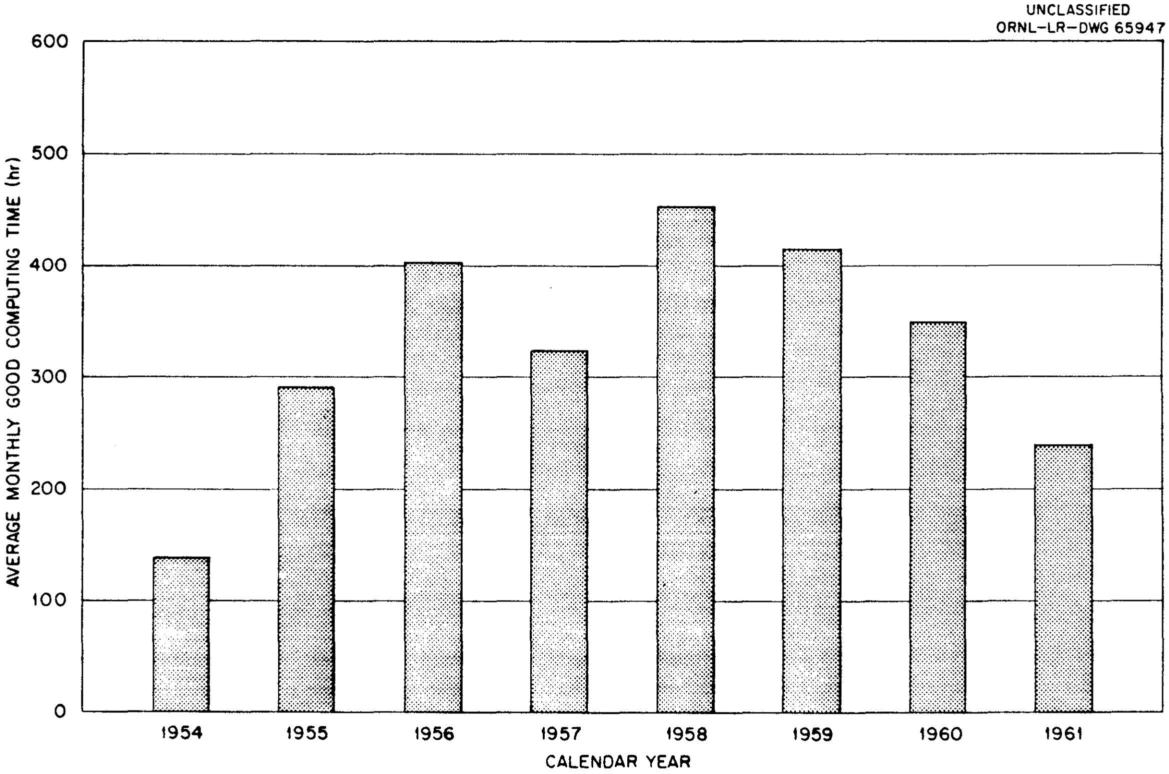


Figure 19—Use of ORACLE since 1954

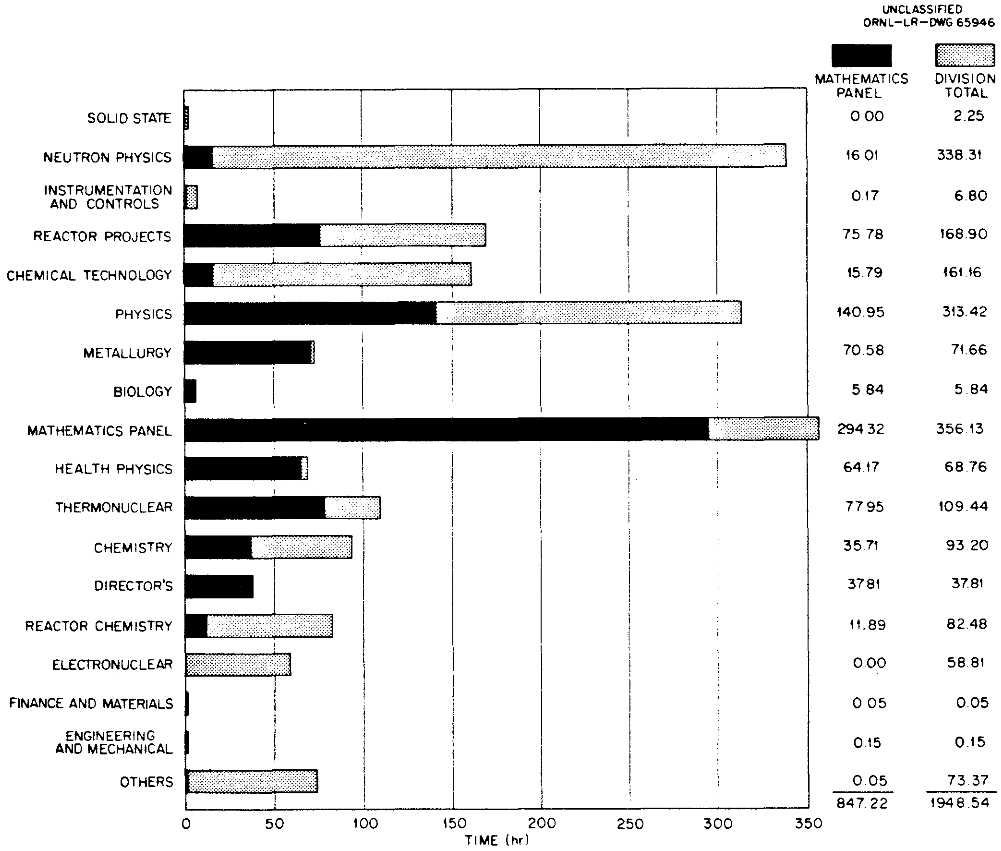


Figure 20—ORACLE time used by divisions during 1961

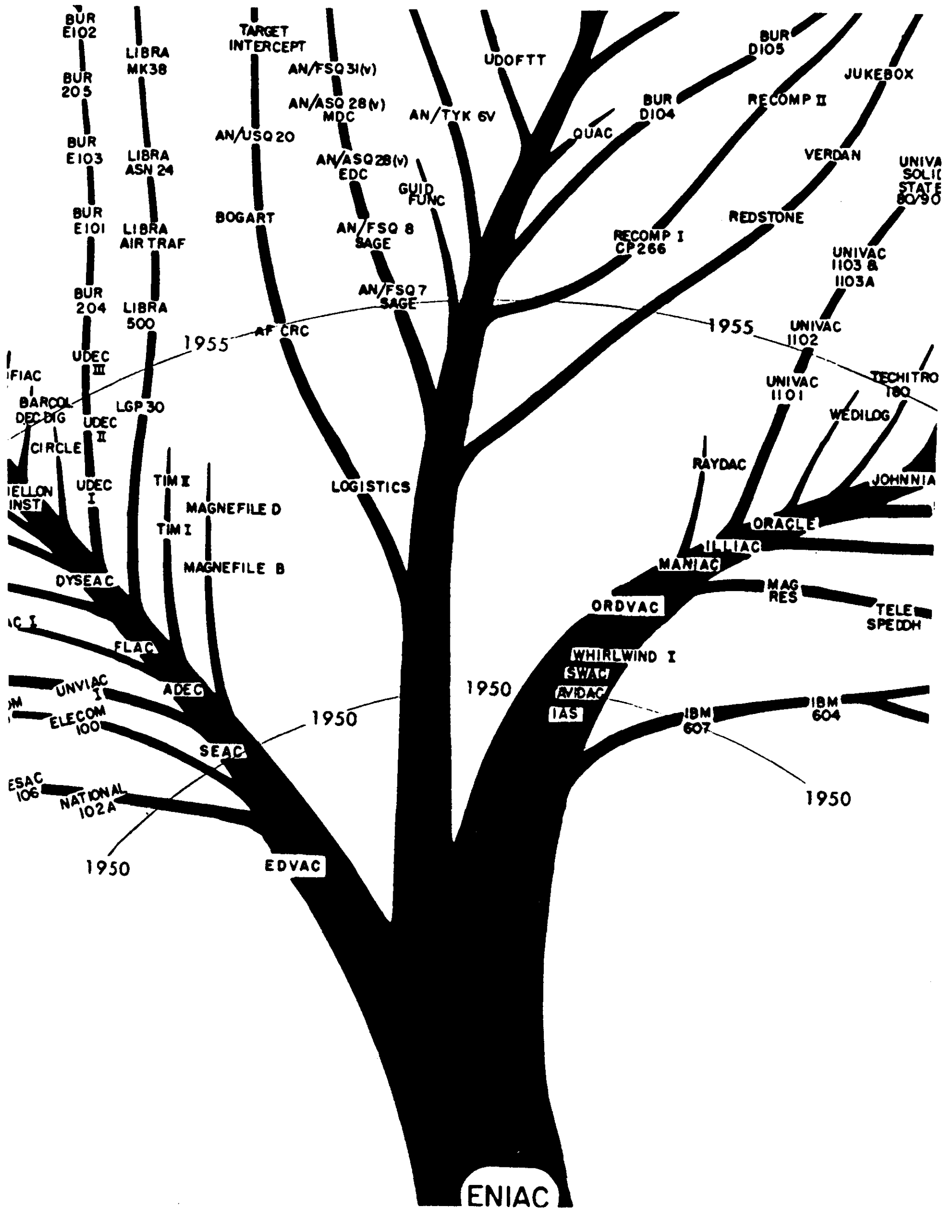


Figure 21—ORACLE's position in computer evolution

1. New engineering approaches should be considered and implemented because there is plenty of latitude for improvement in this young technology;
2. Reliability must be a top priority as a design criterion and during all phases of operation because of the increased user reliance on computers for very important tasks; and
3. A very large potential exists for technology improvements in future generation computers even beyond our fourth and fifth generation machines.

A few of the legacies the ORACLE left are trained people at ORNL (Oak Ridge National Laboratory) in all phases of computer development (hardware, software, operations and administration), thousands of hours of valuable scientific and engineering computing in all scientific disciplines of the laboratory, an enviable record of up-time computing (consistent with tube-type computers), and a cost performance ratio that was an economic scientific bargain for the laboratory.

PIONEER REFLECTIONS

Each new generation of computer people should be aware that knowledge of past computer generation experiences would be valuable in order to build upon and improve future generations of computers. A lot of reinventing of the wheel and repetition of past errors is being committed by newer generation people who are not interested, won't listen and won't take the time to study past experiences.

Williams tubes: A remembrance

by EDWIN L. HUGHES
Melbourne, Florida

ABSTRACT

In keeping with the NCC '85 theme of "Technology's Expanding Horizons," this paper reviews one of the important computer technologies of 35 years ago, the Williams tube. Conceived by Professor F. C. Williams of the University of Manchester in the United Kingdom, it was the first random access memory technology, and was employed by most of the von Neumann computers of the early 1950s. This paper describes its theory of operation and how it was implemented at the University of Illinois.

The theme of the 1985 National Computer Conference is "Technology's Expanding Horizons." That would have been a totally appropriate theme for a conference in 1950, as well, and I would like to look back at an important digital computer technology of approximately 35 years ago, namely, the first random access memory technology.

I joined the University of Illinois research faculty as a member of the Digital Computer Laboratory in September 1949. My first assignment was to help develop the memory for the ORDVAC computer.

The computer industry was in its infancy, and many fundamental technologies were undeveloped. One of the most difficult problems was how to store digital information conveniently for the programmer. At the time I joined the Computer Lab, the only widely used storage technology that I knew about was the mercury delay line. Those delay lines were *serial*, that is, "first-in-first-out." It was obvious that a *random access* memory would make for much quicker computing by avoiding the waiting time associated with the serial delay lines.

At the time I joined the Lab, there were two such developments under way. Jan Rajchman at RCA Research Laboratories in Princeton, New Jersey was at work on an electrostatic storage tube called the *SELECTRON*. The other development, being undertaken by the several von Neumann project teams, was the Williams tube.

F. C. Williams, a professor at the University of Manchester, England, had conceived of a low-cost, high-speed technique for storing binary data. His technique made use of the fact that ordinary glass and the phosphors used in cathode ray tubes (CRTs) have an electron secondary emission ratio greater than 1. When the material is bombarded by an electron beam, the bombarding (primary) beam is absorbed by the material, and additional electrons (called secondaries) are emitted from the surface. If more secondaries are emitted than primaries are absorbed, the surface will be left with a net positive charge; if the spot is bombarded for a long enough time, the positive potential will become great enough to attract some of the secondaries back to the surface, thereby reducing the effective secondary emission ratio to 1. Williams conceived of a technique for establishing, sensing, and erasing those charges, using the extreme agility of the electron beam. This would lead to random access. It was also his thought that commercially available CRTs could be used for computer memories.

Storing binary information requires that the storage system have two stable states. The uncharged spot was one such state, and the positive charge resulting from the bombardment of the screen was the other. The detection of these two states was a system problem of some considerable interest.

The charge state of the spot was inferred by noting the net change in the charge when the spot was bombarded by a "read" pulse. The net change in charge was sensed by an amplifier which was capacitively coupled to the spot. It was in fact connected to a copper screen glued to the outer face of the tube. When the beam struck an uncharged spot, more secondaries were emitted than primaries were absorbed in a ratio of about 1.2 : 1 (the secondary emission ratio), and a positive change in charge was sensed. When a previously charged spot was read, there was no net change in the charge; thus, the signal from the amplifier was merely the derivative of the arrival of the CRT beam at the screen.

Some considerable work and secretiveness went into determining exactly how the charge left by the interrogating beam was erased when it was necessary to restore the spot to an uncharged condition. Williams' original system positioned the beam at a critical distance from the initial location and established a second positive charge. Some of the secondary electrons from that second spot flooded the first spot and effectively erased it; this became known as the *two-dot* technique.

Other systems defocused the beam to discharge the first spot, while still others drew a circle around it. At Illinois, we decided that it was more effective to drag the beam from the read position to the write position. This resulted in a pattern of dots (ones) and dashes (zeroes) on the face of the CRT. The act of moving the beam to create the dash was referred to as *twitching*. The first spot was known as the *read* and the second as the *write*. Some of the secondary electrons freed by the beam during the twitch to the write spot discharged the read spot.

It was recognized immediately that the charges thus established on the face of the CRT were impermanent. To make a practical memory system, it was necessary to restore the contents of each storage location before the charge leaked below a critical threshold. We chose to refresh the memory at least 10 times per second, even though some storage tubes had held their data for up to 90 seconds. To accomplish the refreshing operation, alternate memory cycles were designated *action* and *regenerate*. The regenerate cycle was driven by a free-running counter that simply went through the entire memory array in sequence, read each spot, and restored it to the value it read.

A still larger problem confronted us, for when the beam was "writing" the zero, and its secondary electrons were intentionally discharging the read spot, they were unintentionally discharging nearby read locations. Spacing of the white spots from adjacent read spots was therefore critical to the operation of the memory. Too close spacing resulted in one write spot erasing the wrong read spot; too loose spacing resulted in storing less than the maximum amount of information on the tube. We developed a figure of merit for the Williams tube systems, called the *read-around ratio;* this was the number of times adjacent "dashes" could be read and regenerated without restoring the test spot before the test spot was no longer recognizable as a "dot." The read-around ratio really limited the true random-access nature of the memory for the programmer. One spot could not be accessed too frequently or the surrounding data or instruction would be destroyed.

Early memories simply deflected the beam in a regular rectangular array. We determined that it was better to offset each column by half a space and to twitch up on even-numbered rows and down on the odd-numbered rows in such a way that the ends of the dashes were together and as far away from the other read positions as possible. The spacing of alternate pairs of rows was an operator adjustment, made while the machine ran the read-around-ratio test program. As I recall, that ratio was about 16 : 1 for the ORDVAC at the time we delivered it to the Aberdeen Proving Grounds.

When I arrived at the University, the team had already selected the 3KP1 for the storage tube; most other labs selected 5-inch tubes. We felt that the shorter beam and attendant lesser beam dispersion, together with a better-focusing electron gun, would more than compensate for the smaller diameter of the tube face. Subsequent experience proved our theory correct; we were able to store 1,024 binary digits (bits) on each tube while most other projects were able to store only 256 or 512 bits. However, we had to overcome several practical problems in order to use this commercially available tube.

We purchased our tubes from RCA. At first we had fairly good success with them, but suddenly we received several that simply would not work. After several frustrating experiments, we discovered that the charges were leaking off much more quickly than on previous batches of tubes. We discussed the matter with the engineers at RCA and learned that the envelopes of 3KP1 tubes were made from two types of glass, lime and lead. The first tubes we had received had, happily, been made with lime glass, which has a significantly higher resistivity than lead glass. Henceforth, we specified only lime glass for the envelopes of the tubes, a significant deviation from the "commercial grade" objective.

Early in the project, we also learned that there were flaws in the tubes that made certain small areas unsuitable for use; they simply refused to accept a charge. I wrote my M.S. thesis about those flaws and designed a fixture that would allow me to photograph (*map*) the inner surface of the tube face to determine the size and density of flaws. If the flaws were small in size and number, we could position the spots on the tube face to dodge them; if they were too large or numerous, we simply had to set that tube aside as unsuitable for our use.

Some laboratories, IBM for one, zapped the external face of the tube with a Tesla coil and observed that the flaws would disappear. They were presumably propelled from the surface by the intense electrostatic force. I do not recall that we used that technique.

The University of Illinois had a vacuum tube laboratory that helped us in our research. Together, we discovered that the flaws were for the most part small flakes of graphite from the aquadag lining of the envelope. CRTs were lined with aquadag as an electrode to collect the stray electrons, principally the secondaries from the screen. Therefore, the coating was essential to the operation of the memory tube.

The electron gun structure was supported on spring fingers that scraped little bits of the coating off the tube wall when the gun was inserted into the envelope. Of course, it was much more convenient for the assembly-line worker to set the envelope face down on the work bench in order to push the gun into it. This resulted in carbon particles falling down onto the inside face of the tube.

RCA agreed to assemble the 3KP1s for memory use "face up" only, and with extreme care. RCA also designed a special package intended to keep the tube face up during shipment (another deviation from standard commerical practice). In fact, RCA later produced the 3VP1 tube, which was specifically intended as a Williams tube.

The ancillary systems were also very important to the successful development of the system. High storage density required very precise location of the charge spots on the face of the CRT. Our laboratory—in the old Electrical Engineering Building, which still housed the D-C machines laboratory—was full of fluctuating magnetic fields. It was necessary to protect the CRTs from those fields, so each tube was housed in a four-layer metal box made of alternating layers of copper and Mu-metal, with the copper on the inside.

The deflection circuits for positioning the beam were equally critical. Highly regulated and filtered power supplies were used, along with digital-to-analog converters using the best vacuum tubes and resistors available. The CRT accelerating voltage was 2,050 volts, with the positive (screen) terminal at +150 volts. This meant that the filament transformers had −1,900 volts on them, and had to be of the highest insulation quality to keep arcing or corona from injecting noise into the system.

The actual storage mechanism was a subject for debate among the several project teams at first. The first question was that of the role played by the phosphor. This question was settled at the University of Illinois. We had the vacuum tube lab make us a few tubes with no phosphor, and we showed that they worked just as well as (but no better than) the standard tube with a phosphor.

Another question arose over whether there were any long-term changes in the CRT inner surface due to the bombardment with the electron beam pattern for protracted periods of time. The photographs I took for my master's thesis clearly showed the storage raster patterns on previously used tubes, even though several weeks had passed since the tubes had been used for data storage. As far as I know, no one ever detected any deleterious effects of this permanent change.

The "flaw scanner" also permitted some inadvertent industrial espionage. Arthur Samuel was at one time the head of the University Vacuum Tube Laboratory. He subsequently left the Lab to work for IBM. When he heard of my fixture to scan the inner surface of the CRTs, he sent in some tubes to be scanned. He had hoped to learn something about the quality of the tubes. I sent back photographs that showed the exact secret raster pattern of the IBM memory!

There were several computers built with Professor Williams' invention: von Neumann's machine at the Institute for Advanced Study; ORDVAC and ILLIAC at the University of Illinois; AVIDAC and ORACLE at Argonne National Laboratory; SEAC and SWAC by the National Bureau of Standards; MANIAC at the Los Alamos National Laboratory; and the IBM Defense Calculator, the forerunner of the IBM 701 and 702. There may have been others.

The Williams' tube was a considerable contribution to the computer technology of its day, but its day was short-lived. It was soon supplanted by the invention by Jay Forrester (MIT) of the ferrite core memory, which has in turn been made obsolete by the semiconductor memories. Taken in the context of today's personal computers, the University of Illinois Williams tube memory contained 40 × 1,024 bits, or about 5K bytes. Not much by today's standards, but enough to make possible the brilliant work of hundreds of scientists, mathematicians, engineers, and programmers for several years.

SUGGESTED READINGS

1. Bland, G. F. "A Storage System for an Electronic Digital Computer." Master's thesis, University of Illinois—Urbana, 1950.
2. Eckert, J. P., H. Lukoff, and G. Smoliar. "A Dynamically Regenerated Electrostatic Memory System." *Proceedings of the Institute of Radio Engineers,* Vol. 38, (1950), p. 498.
3. Hughes, E. L. "A Circuit for Studying the Secondary Emission Characteristics of Cathode Ray Tube Phosphors." Master's thesis, University of Illinois—Urbana, 1950.
4. Hughes, E. L. "General Description of the ORDVAC Memory." Paper presented at National Bureau of Standards Symposium on Williams' Electrostatic Storage, Washington, D.C., December 1951.
5. Mutter, W. E. "Improved Cathode-ray Tube for Application in Williams Memory System." Paper presented at the American Institute of Electrical Engineering Winter General Meeting, New York, January 1952.
6. *ORDVAC Manual.* University of Illinois—Urbana, October 1951.
7. Parker, C. V. "Charge Storage in Cathode-ray Tubes." *Proceedings of the Institute of Radio Engineers,* August 1951.
8. Williams, F. C., and T. Kilburn. "A Storage System for Use with Binary-digital Computing Machines." *Journal of the IEEE,* Part IIIA, 96, (1949), p. 81.

Post-AVIDAC computer technology at Argonne National Laboratory

by JAMES W. BUTLER
Butler Associates
Lemont, Illinois

ABSTRACT

After the development and engineering of AVIDAC and ORACLE, the "first generation" machines, had been essentially completed, computer systems research at Argonne National Laboratory evolved over a period of time in several directions. An attempt is made to trace the particular thread of this research which encompassed work in general-purpose computer architecture and in image processing and pattern recognition. Whenever possible, assertions are supported by reference to original written sources.

INTRODUCTION

After the development and engineering of AVIDAC and ORACLE, the "first generation" machines, had been essentially completed, computer systems research at Argonne National Laboratory evolved in several directions, driven in general by the interests of the staff and by certain environmental forces.

Coalescence of the applied mathematics and computer-related activities in the Physics Division led to the formation of the Applied Mathematics Division (AMD) in 1956, furnishing an interdisciplinary organizational structure within which computer development could proceed.

The laboratory culture afforded a good selection of interesting and significant problems that appeared to be solvable by computer techniques, and moral and financial support was available directly from the Atomic Energy Commission (AEC) and indirectly through disciplinary clusters engaged in research and development in nuclear reactor engineering, nuclear physics, high-energy physics, chemistry, and biology.

Perhaps more important, the successful development of the AVIDAC and the ORACLE engendered an enthusiastic attitude among the Argonne staff and added an indispensable element of confidence that such projects could actually be accomplished.

This paper is not a comprehensive review; the subjects treated are only those of which I have some personal knowledge. The review article by Margaret Butler[1] provides some additional information on the subjects discussed and covers a number of topics that are not included elsewhere.

GEORGE[2]

The following excerpts are from a press release issued by Argonne National Laboratory in November, 1957.

> An electronic computer that runs on electricity produced by atomic power, and in turn helps scientists learn new facts on how to produce atomic power better, now is in operation at Argonne National Laboratory, Lemont, Illinois.
>
> Built and designed by Argonne, the new computer has been christened "GEORGE", so Argonne mathematicians will be literally letting GEORGE do it when they present this new machine with problems to solve.
>
> Since Argonne generates much of its own Laboratory electricity with its Experimental Boiling Water Reactor, GEORGE may well be one of the first electronic computers to use atomic power.

This press release reveals one possible motivation for the choice of the name GEORGE. Alternatively, Jean Hall writes, "... its initials do not stand for anything, it doesn't end in AC and we believed the machine would be 'real george.' "[3] I have not been able to trace the provenance of this expression; perhaps it can be translated into newspeak as *cool!, to the max!,* or *bad!*

GEORGE was actually the third general-purpose computer built at Argonne, but only the second one built for local use. Design work was started in mid-1954 and the machine attained operational status in 1957, as noted in the above press release. Out-of-pocket construction cost was reported at slightly over $300,000. The system performed useful work until it was decommissioned on April 11, 1966[4] along with the GUS (George Unified System) computer complex, of which it had become a part.

Approximately 3500 vacuum tubes and 3000 "crystals" (diodes) were used in the logic and control sections of the machine. The main store consisted of a 4096-word magnetic core memory, and auxiliary storage was provided in the form of a four-unit magnetic tape bank with a capacity of 4,000,000 words. As in the AVIDAC, a 40-bit word length was used, while the accumulator and quotient registers were 41 bits wide to allow out-of-range results to be detected. Memory cycle time was 27 microseconds, significantly slower than the design goal of six to seven microseconds.

The auxiliary-storage tape drives[5] were physically the same units that had been designed for the AVIDAC, using two-inch-wide plastic tape with 42 recording channels across. The design called for a burst transfer rate of approximately 5000 words/second, and the control provided for forward or backward searching for data blocks concurrently with computation.

From our present perspective, the most interesting aspect of the GEORGE architecture is the order structure, which was refined in a series of informal and often lively meetings that were presided over by J. C. Chu and the minutes of which have apparently been lost. My role in these proceedings was a minor one; as I recall, the major players were Chu, Jean Hall, and Herb Gray. The resulting instruction code could cause several operations to be performed by one 40-bit instruction word, and lent itself to reasonably easy programming in absolute hexadecimal form. A number of system and application programs were in fact coded in this manner before the relocating assembler GAR (written by Herb Gray) became available.

Representing the GEORGE instruction word as 10 hexadecimal "digits" (h0...h9), the components of the order code are the B-address (h0h1h2), Order (h3h4), Tag (h5h6), and A-address (h7h8h9), mnemonicized as "BOTA." GEORGE programmers could often be heard muttering this incantation as they went about their work. The two-address instruction code allows for computation of the effective address by adding to the A-address the contents of the location pointed to by the B-address, in effect implementing 4096 index registers or "B-boxes" (a British import). The arithmetic instructions incorporate options for dual-precision (19- or 39-bit) arithmetic, a feature which was included specifically for use in programming monte-carlo problems.

As a typical example of the instruction-code capability, we can construct a loop-terminating instruction as follows. Assume that the loop entry is at hex 100 and that the B-segment of the word at hex 300 has been initialized to the negative of the iteration count. The instruction 300 72 4C 100 would, if placed at the end of the loop, increment the count at 300, jump back to 100 if it is still negative, or fall through to the next instruction when the count reaches zero. If GEORGE still existed, it would be a good machine on which to play the game *Core War,*[6] which is currently in vogue among hackers, because the two-address order structure is similar to the instruction format of the *Redcode* interpreter commonly used to program such games.

Aside from the 40-bit word length and method of using magnetic tape for auxiliary storage, the influence of the AVIDAC development on the design of GEORGE is not visible by a surface examination. However, many of the same people worked on both machines and used their experience with AVIDAC to engineer many of the internal details having to do with machine logic and general principles of construction.

GEORGE-FLIP AND GUS[7]

In 1957 or thereabout, as the hardware engineering of GEORGE was being completed, it became evident that a significant shortcoming of the machine was its lack of a hardware floating-point capability.[8] Work began on the design of a floating-point arithmetic unit to remedy this perceived deficiency.[9] This scheme must be counted as at least one of the earliest uses of this type of organization. Today, of course, practically every minicomputer or microcomputer is designed to accommodate an optional floating point unit (FPU) or co-processor chip, albeit on a scale of ounces or grams rather than, as it turned out, tons.

During that time period, the accuracy of floating-point computations was an active area of research at ANL and elsewhere, which led to a proposal to calculate by hardware the result of a floating-point operation, and an index of the precision (number of significant bits) to be associated with the result.[10] Moreover, the availability of reliable transistors allowed the logic circuitry to be implemented with all solid-state components[11] instead of the slower and presumably less secure vacuum-tube elements used in GEORGE.

The fusion of these ideas resulted in the design and construction of a processing unit (FLIP) to assume most of the computational functions of the combined system, delegating input-output control and certain switching operations to the GEORGE component. The GEORGE-FLIP system, later known as GUS, achieved operational status in 1964 following the addition of two 4096-word (80 bit, two-microsecond word cycle) core memory modules which had been developed under a contract with New York University [Press release #479, Public Information Office, Argonne National Laboratory, January 8, 1964]. The system remained under more or less continuous development until it was dismantled in 1966. It is not clear from the available written records that the FLIP processor was actually used in production codes run on the combined system; together with other components of the GUS system, it served more as a focus for research in computer architecture and the engineering of multicomputer systems.

The architecture of FLIP as realized in its instruction set is not without interest today, as it incorporates several features that are recognizable in present-day computers. While the order code describes an architecture which does not fit well into either of the now fashionable classifications, CISC (complex instruction set computer) or RISC (reduced instruction set computer), the idea is rather to define only a few instructions and make each one complex enough to perform a number of related operations.

Floating-point operands are full 80-bit words; however, all integer arithmetic and indexing instructions operate on 20-bit subwords called *control groups* if they are part of a memory word, or *bins* if they are implemented as hardware registers. Because the register control groups can serve as index registers, the name *bin* is used to avoid semantic confusion with the floating-point index of significance. Integer and indexing instructions interpret memory as consisting of 20-bit "words." Carrying the design philosophy used in GEORGE even a bit further, one integer instruction (consisting of from one to four control groups) can perform a remarkable number of different operations.

Stan Hansen writes:

> The orders for the handling of integers are of a generalized form X[+ | −]Y → TEST,Z with the ability to jump on the results of the test. X may be one of the 16 bins either cleared or held. Y may be one of the 16 bins, an address used directly, the contents of an address, the power or index of a number in the universal floating point register, the jump register which contains the location of the last jump order, the location counter which contains the absolute location of the present command, or various often used constants. The results of the operation may be stored back in one of the bins, in the memory, or in the power or index sections of a floating point number.[12]

Although the instruction set allows absolute addressing, the preferred algorithm is relative to the location counter with the contents of up to four bins added to form the effective address. With a suitable assembler, this architecture would then allow segmented programs to be written by designating base registers somewhat after the manner of the IBM 360 or the Intel 8086/8088 systems used in various desktop computers including the IBM Personal Computer. The FLARE assembler[13] in fact implements a hierarchic block structure managed by creating an accessory symbol table of block names.

Floating-point operands are of the following form: (one-bit control flag, six-bit index of significance, nine-bit signed exponent, 62-bit signed fraction), while the generalized floating-point instruction may be written as

$$(\mathrm{aR}[\,+\,|\,-\,]\mathrm{A})[\,+\,|\,-\,|\,\times\,|/|\,\text{geometric-mean}]([\,+\,|\,-\,|\,+\,\mathrm{abs}\,|\,-\,\mathrm{abs}]\mathrm{B}) + \mathrm{aC} \rightarrow \mathrm{R,C}$$

where a = [1|0], R is the universal arithmetic register, and A, B, and C are memory or register operands. I played a role in promoting the use of the binary geometric mean operator rather than the less general unary square root. The algorithm for updating the index of significance may be found by consulting the original works already referenced.

In retrospect, it might be said that the successful engineering of a complex system such as GUS was possibly beyond the capabilities of the small group constituting the Applied Mathematics Division, although many good ideas were certainly advanced and implemented. It is nonetheless interesting to speculate on the course that events might have taken in the absence of the incipient explosive growth in commercial computer development.[14]

IMAGE PROCESSING AND PATTERN RECOGNITION

Image processing research at Argonne began in 1961 with the development in AMD of the CHLOE film scanner,[15] and persisted in some form until the shut-down of ALICE, its successor machine, in 1984.[16] This program, which must be considered to be a conspicuous success, contributed to divers scientific and technical disciplines including high-energy physics, cell biology, bone autoradiography, chemistry, materials science, fingerprint identification, land management, and computer technology. From about 1970 on, general image-processing research gradually declined because of increasing pressure from funding centers to shift the program into a production mode.

CHLOE[17]

Following the invention of the bubble chamber by Donald A. Glaser in 1952, high-energy physicists began to think seriously about using computer-assisted techniques to cope with the potential flood of pictorial data producible by these devices in conjunction with the new accelerators coming into operation in about 1955.[18] In subsequent years, physics groups in a number of countries attacked this problem by developing an assortment of both mechanical and electronic measuring devices, generating voluminous amounts of literature in the process, much of it in the form of conference proceedings such as the one covering the Cavendish Laboratory Conference in 1970.[19] M. K. Butler and I wrote a brief summary of the situation as it existed in 1966.[20]

Because a typical bubble-chamber photograph shows many charged-particle tracks and contains a large amount of mostly extraneous information, it proved infeasible to attempt complete automation of the measuring process for these pictures. The spark chamber[21] was introduced around 1960 as an alternative track-visualization device which promised to generate photographic data in a form more accessible to automatic measurement, mainly because it tends to produce a large number of simple images rather than a smaller number of complex ones.

The active component of a spark chamber consists of a series of equally-spaced parallel metal plates or grids held at alternating high and low electrical potentials of sufficient magnitude to produce a near-breakdown field strength between the plates. Under the proper conditions, an energetic charged particle passing through the plates will ionize the gas in the gaps and cause visible sparks to fire along the particle path. Cameras aimed between the plates can be used to record stereoscopic views on film (usually 35mm), thereby in principle allowing the particle track to be reconstructed in three dimensions.

Design of the CHLOE film scanner began in 1961, and the system became operational in 1963. Substantial financial support and encouragement were provided by the recently-organized Argonne High Energy Physics Division. While the specific design goal was to fashion a device capable of automatically measuring and interpreting data from 35mm spark-chamber film images, the system (wisely, as it turned out) was made as flexible as possible to accommodate other possible types of photographic data.

The system was built around an ASI-210 minicomputer* as the central control element, and included a scanner control unit with two 35mm film scanning stations attached, a console typewriter, a photoelectric paper-tape reader, a paper-tape punch, a slaved monitor display, and a magnetic tape drive. A bidirectional communication link was also set up to enable data transfer between the CHLOE system and the CDC 3600 central computer then operated by the Applied Mathematics Division.

The ASI-210, quite a respectable computer for its day, used a 21-bit word length with solid-state binary logic, and included an 8192-word core memory with a two-microsecond word cycle time. External device control was accomplished via a 21-bit external "bus," six bits of which made up the device address. Thus, a maximum of 64 devices could be accommodated. Interrupt response was implemented by provision of 64 interrupt-vector locations (mapped to the 64 device addresses) serviced in round-robin sequence by an interrupt scanner. Two "cycle-stealing" buffered input/output channels were provided to handle block data transfers to and from devices.

All programs for the ASI-210 were written in Assembler language, at first with a two-pass paper-tape assembler supplied by the manufacturer, and later with a cross-assembler written by Herb Gray to run on the CDC 3600.[22]

The scan controller using control logic based on the NOR module designed for use in the GEORGE-FLIP system,[23] and scanning stations were built by AMD engineers. The controller was first built with vacuum-tube circuits which were later replaced by transistor modules. The cathode-ray-tube scanner at each station was driven by two 12-bit X and Y counters, with starting and ending values set by four bounds registers. The spot was not actually moved, but was turned on at one location for one microsecond after which it was blanked for three microseconds while the deflection was updated to refer to the next location to be examined. Perhaps the most inter-

* By the Advanced Scientific Instruments Division of Electro-Mechanical Research, Inc., in 1963.

esting feature of the scan controller logic is that it implements run-length image encoding by default, in that a coordinate is only transmitted to the computer when a significant change in transmitted light intensity is observed. Although the transmitted light intensity was measured by an eight-level discriminator, only two-level (light-dark) density measurement turned out to be useful in most situations, since not much was known at the time about setting discriminator levels. The scanning tube beam current was later placed under program control (1024 levels) to partially alleviate this problem.

The first film type attempted came from a spark-chamber experiment carried out at CERN by a high-energy physics group associated with Arthur Roberts. Due to the general layout of the experiment and the photographic optics used, this film turned out to be very difficult to measure, and in spite of a rather large expenditure of effort, the project was never brought to a production stage. However, enough was learned in the attempt so that film from several later spark-chamber experiments was measured successfully. Shortly thereafter, interest in spark chambers began to wane as other track measuring devices such as wire chambers capable of producing data without an intervening photographic process were invented. The discovery of more exotic particles also caused high-energy physicists to turn increasingly to bubble chambers because of the much greater attainable precision of measurement. Don Hodges and the AMD engineering group then began development of a family of high-precision interactive scanning machines, known generically as POLLY, for measuring bubble-chamber photographs.[24] These became popular and were known by other related names such as DOLLY, MOLLY, etc.[25]

CHLOE was used with considerable success in the other scientific fields mentioned such as biology, and material science.

Quite a bit of effort was expended on the so-called *chromosome-pairing* or *karyotyping* problem in cytogenetics, a branch of cell biology.[26] In a tissue culture containing dividing cells, treatment with drugs such as colchicine can arrest the process of mitotic division at the stage called *metaphase* in which the individual chromosomes with proper staining are visible through an optical microscope. Microphotographs of the chromosome complements of individual cells, called *spreads,* served as input for processing by the CHLOE film scanner. Due to the nature of the biological process, the chromosome images in the spread occur in pairs; the pattern recognition problem consists of sorting the chromosomes into pairs, the sorted aggregate being called a *karyotype.* Karyotyping has value, for example, in effecting prenatal diagnosis of birth defects such as Down's Syndrome, and is still used in genetic counseling. This problem was very popular (at one point I counted 13 different institutions engaged in solving it), and was in fact essentially solved in a technological sense by our group and some others. However, none of the systems that were developed ever reached production status, due in part to the differing world views of the "dry" and "wet" scientific types involved.

Another biological project of some interest was a study of bone autoradiographs undertaken in collaboration with Elizabeth Lloyd and John H. Marshall of the Argonne Radiological Physics Division.[27] One objective of this research was to study the role of bone in the homeostatic regulation of calcium ion concentration in the blood plasma. The procedure involved injecting a radioactive tracer such as calcium 45 and later obtaining a small section of bone and placing it in contact with a photographic plate for a suitable time (often several months). By modifying programs previously used in the chromosome studies, it proved to be feasible to use the CHLOE system to scan these plates and measure the optical density at a large number of points distributed over the bone section. The numerical data thus obtained were used to assess the total radioactive dose to the bone as well as the dose distribution over the area of the section.

A notable application in materials science was the work on fragmentation studies carried out with D. Armstrong and W. Gunther of the Argonne Reactor Analysis & Safety Division.[28] In these experiments, single drops of materials such as molten tin, lead, bismuth, and uranium dioxide are dropped into candidate nuclear reactor coolants such as water and sodium. In the CHLOE project, using tin and water as the experimental materials, the metal fragments were recovered and photographed on a light table after arrangement by hand to eliminate overlaps. The film was then processed by the scanner to locate the centroids of the individual particles and measure shape coordinates for computing characteristic parameters such as particle size distribution, Sauter mean diameter, average particle thickness, and total two-dimensional surface area.

Falling somewhat outside the usual areas of scientific research, a project on fingerprint identification and classification was begun in about 1965 by Ben Shelman[29] and carried on for some time thereafter. The object of this work was to develop an automatic fingerprint identification and filing system based partly on the classical 10-finger Henry System, but aimed more toward facilitating single-print identifications by locating and recording the small features called *minutia.*

In early 1970, it became increasingly difficult to keep the ASI-210 in running order due to progressive failure of the core-memory components caused by the high temperature inside the oven housing the memory (a unique feature of this machine). The CHLOE system was shut down on April 1, 1970[30] and effort was concentrated on completing construction of the ALICE system.

A valuable lesson which should have been learned during this period was expressed by Albert Crewe (then Laboratory Director) in 1963 [Press Release ANL-P10-451, Public Information Office, Argonne National Laboratory, 17 July 1963].

> The fact that three Argonne scientific divisions cooperated in the application of the CHLOE system to experiments in Biology and High-Energy Physics also demonstrates the value of an interdisciplinary approach to scientific investigation.

ALICE[31]

The ALICE image processing system, the successor to CHLOE, became operational in 1970 and was more or less complete by Spring, 1972. Compared to CHLOE, the system incorporated a larger and faster computer, a much more sophisticated controller, and a number of scanning stations capable of accepting a wider variety of input media. Other improvements were the ability to measure 64 optical-density levels, and the use of a more flexible display monitor with interactive capability.

The computer component of the system was a Digital Equipment Corporation (DEC) PDP-10 with a core memory of 32,768 36-bit words. Standard peripheral equipment included a Model 33 Teletype console typewriter, card reader, line printer, paper-tape input and output, three DECtape units, and both seven- and nine-track magnetic tape drives. System software included the PDP-10 Single-User Monitor (!), a full complement of utility programs, and a Fortran compiler. Additional system software was written locally to allow easy programming of scanner applications via Fortran subroutine calls.[32]

The final configuration included four scanning stations, station 0 for 16mm or 35mm sprocketed film, station 1 for 35mm film, station 2 incorporating a Leitz Orthoplan research microscope, and station 3 for use with opaque media to be scanned by reflected light.

From an architectural standpoint, the most interesting feature of the system was the controller. Instead of using X and Y counters as in CHLOE, the spot deflection was driven by a display file[33] so that arbitrarily complex scan patterns could be generated, a feature which was thought to be promising for future research in pattern recognition. Another interesting element of the controller was a digital circuit which could be loaded with parameters from the computer to dynamically correct pincushion and edge defocus distortion.[34]

For the most part, the projects carried out with the ALICE system represented extensions of work begun using CHLOE, such as the chromosome analysis research and further development of the fingerprint classification system. After 1975, the system was used almost exclusively to process soil maps for an agency of the State of Illinois, until it was finally shut down on September 28, 1984. One notes that this was the last of the Argonne-built computing devices.

In my view, ALICE was a very advanced and versatile machine, and I am not aware that its design has been surpassed by any more recent systems. It would be interesting to build a similar system using modern microprocessor and display control technology.

REFERENCES

1. Butler, M. K. "Argonne National Laboratory." *Encyclopedia of Computer Science and Technology,* pp. 141–179. New York: Marcel Dekker, 1975.
2. Kassel, L. and Donald A. Flanders. *GEORGE Programming Manual,* ANL–5995. Argonne National Laboratory, May 1959.
3. Hall, J. F. "Introducing GEORGE." Unpublished memorandum, Argonne National Laboratory, February 1956.
4. Harman, H. B. (ed.) "Exit GEORGE." *The Argonne News,* 15-11, Argonne National Laboratory, May 1966.
5. Merrill, L. "40 Channel Magnetic Tape Auxiliary Memory." *Proceedings of a Symposium on Large Scale Digital Computing Machines,* ANL–5181. Argonne, Ill.: Argonne National Laboratory, 1953.
6. Dewdney, A. K. "Computer Recreations." *Scientific American,* May 1984, pp. 14–22, and March 1985, pp. 14–23.
7. Miller, W. F. and R. Aschenbrenner. "The GUS Multicomputer System." *IEEE Transactions,* EC–12 (1963), pp. 671–676.
8. Miller, W. F. "Mathematics and Computer Research at the Argonne National Laboratory." Unpublished memorandum, Argonne National Laboratory, April 1961.
9. Jacobsohn, D. "Preliminary Discussion of Floating Point GEORGE," Unpublished memorandum, Argonne National Laboratory, April 1957; and *AMD Technical Memorandum No. 52,* Argonne National Laboratory, July 1963.
10. Gray, H. L. and Charles Harrison, Jr. "Normalized Floating-point Arithmetic with an Index of Significance." *Proceedings of the Eastern Joint Computer Conference,* Boston, December 1–3, 1959, pp. 244–248.
11. Salter, F. "Circuitry of FLIP." *AMD Technical Memorandum No. 11,* Argonne National Laboratory, July 1960.
12. Hanson, S. C. "The System Organization of FLIP." *AMD Technical Memorandum No. 12,* Argonne National Laboratory, July 1960.
13. Fisherkeller, M. A. and G. A. Robinson, Jr. "ARGUS I, A Programming System for GUS." *AMD Technical Memorandum No. 41,* Argonne National Laboratory, July 1963.
14. Bell, C. Gordon and A. Newell. *Computer Structures: Readings and Examples.* New York: McGraw-Hill, 1971.
15. Butler, J. W., D. Hodges, and R. Royston. "CHLOE, A System for the Automatic Handling of Spark Pictures." *AMD Technical Memorandum No. 35,* Argonne National Laboratory, November 1962.
16. Heiberger, A. (ed.) "ALICE Image Processing System to be Shut Down." *Argonne Computing Newsletter,* 15-9, Argonne National Laboratory, September 1984.
17. Hodges, D. "CHLOE, An Automatic Film Scanning Equipment, Hardware Reference Manual." *AMD Technical Memorandum No. 61,* Argonne National Laboratory, November 1963.
18. Macleod, G. R. and J. Snow (eds.) *International Meeting on Instruments for the Evaluation of Photographs, Summary of Proceedings.* CERN 58-24, European Organization for Nuclear Research, Geneva, October 1958.
19. Lord, D. H. and B. W. Powell (eds.) *Proceedings of the International Conference on Data Handling Systems in High-Energy Physics* (at Cavendish Laboratory, Cambridge, England, March 1970). CERN 70–21, European Organization for Nuclear Research, Geneva, July 1970.
20. Butler, J. W. and M. K. Butler. "Computer Analysis of Photographic Images." *Nucleonics,* 25-2 (1967), pp. 44–51.
21. Roberts, A. "Note on the Problem of Automatic Scanning of Spark Chamber Photographs." *Review of Scientific Instruments,* 32 (1961), p. 531.
22. Gray, H. "The CHLOE Assembler on the CDC-3600: TCA 3600," unpublished memorandum (Argonne National Laboratory, April, 1965).
23. Salter, F. O. "The NOR Circuit and Some of Its Applications." *AMD Technical Memorandum No. 26,* Argonne National Laboratory, December 1961.
24. Barr, R., R. Clark, D. Hodges, J. Loken, W. Manner, B. Musgrave, P. Pennock, R. Royston, and R. Wehman. "POLLY I: An Operator-assisted Bubble Chamber Film Measuring System." *Reviews of Scientific Instruments,* 39 (1968), pp. 1556–1562.
25. Kosner, C. (ed.) *Proceedings of the Informal Colloquium on POLLY and POLLY-like Devices,* ANL–7934. Argonne, Ill.: Argonne National Laboratory, January 1972.
26. Butler, J. W., M. K. Butler, and Barbara Marczynska. "Automatic Processing of 835 Marmoset Spreads." *Annals of the New York Academy of Sciences,* 157 (1969), pp. 424–437.
27. Butler, J. W. "Automatic Analysis of Bone Autoradiographs" In *Pictorial Pattern Recognition.* Washington, D.C.: Thompson Book Company, 1968, pp. 75–85.
28. Butler, M. K. "Application of an Image Processing System to Fragmentation Studies." *Proceedings of a Conference on the Effective Use of Computers in the Nuclear Industry,* CONF–690401. Division of Technical Information, U.S. Atomic Energy Commission, April 1969.
29. Shelman, C. B. "Machine Classification of Fingerprints." *Argonne National Laboratory Review,* 5-1 (1969), pp. 21–33.
30. Anonymous. "In Memoriam: CHLOE 1961–1970." *ALICE Note No. 8,* Argonne National Laboratory, June 1971.
31. Butler, J. W., J. R. Haasl, D. Hodges, B. Kroupa, C. B. Shelman, and R. H. Wehman. "Alice Reference Manual." *AMD Technical Memorandum No. 231,* Argonne National Laboratory, August 1972.
32. Butler, J. W. "FORTRAN and Assembler Level Programming of the ALICE Film Scanner under Monitor Control." *AMD Technical Memorandum No. 212,* Argonne National Laboratory, February 1972.
33. Newman, W. M. and R. F. Sproull. *Principles of Interactive Computer Graphics.* New York: McGraw-Hill, 1973, chap. 5.
34. Barr, R. and D. Jacobsohn. "Digital Hardwired Pincushion and Dynamic Focus Correction." *AMD Technical Memorandum No. 204,* Argonne National Laboratory, October 1970.

Early papers on the ORDVAC and the ILLIAC

Introduction *by* JAMES E. ROBERTSON
University of Illinois—Urbana
Urbana, Illinois

A number of papers, prepared during the time the ORDVAC and the ILLIAC were in operation, are reprinted here. The papers are of interest not only for their factual content, but also because they reflect the attitudes, the areas of concern, and the terminology of the early years, which often differ markedly from those of today.

The characteristics of the ORDVAC are given in the 1952 paper by Meagher and Nash.[1] During the period between completion of the ORDVAC and the ILLIAC—almost a year—the experience gained from operation of the ORDVAC prior to its shipment to Aberdeen Proving Ground resulted in design changes of the ILLIAC. Other changes were suggested by David Wheeler, who arrived from Cambridge University shortly before the ORDVAC was completed. The number of bits assigned to specify an instruction was reduced from nine to eight, for better compatibility with a hexadecimal input. The memory circuits were also improved. Detailed specifications, such as the operation times in the ORDVAC paper, are a snapshot of conditions at the time the material was written and are subject to frequent changes in the early years of a computer's life.

In the early 1950s the need to achieve reliability affected every aspect of the design and operation of computers. Components were tested to ensure that they met specifications, circuits were designed to operate correctly under worst-case variations of component characteristics, and diagnostic programs were not only prepared to provide data to locate faults, but were also intended to detect malfunctions before user programs were affected. Careful logs were kept, and information about malfunctions was recorded at the time of occurrence. The 1953 paper by Wheeler and Robertson[2] describes, among others, the diagnostic program called the leapfrog, which rapidly became the primary test to indicate whether the ILLIAC was sufficiently reliable for service use.

The 1953 paper by Williams[3] reviews the operating experience with the ORDVAC after its installation at Aberdeen Proving Ground. This paper also indicates that a faster paper tape reader was installed, that card-handling equipment was added, and that improvements were made in the electrostatic memory.

The ILLIAC electrostatic memory, with a capacity of 1024 words, was later augmented by a magnetic drum memory, with a capacity of 12,800 words. Although the basic instructions involved a single word transfer, in practice a block of words was transferred under program control. The timing was such that this program was optimized if a transfer was requested at every fifth word recorded on the drum surface. The 1956 paper by Robertson[4] describes the method of counting employed for this purpose. The paper initially describes the binary counter designed for the Institute for Advanced Study computer and copied for the ORDVAC and for the ILLIAC, and indicates that an up counter can be converted to a down counter by a simple rearrangement of wiring. It is next observed that $5 \times 13 = 65$, and that $65 \equiv 1$ (mod. 64), so that a 6-bit counter that advances by 13 for each received count will advance by 1 for each 5 received counts. Finally, 13 can be represented as $16 - 4 + 1$, and advancing by 13 is achieved by simultaneously incrementing at digital positions of weights 1 and 16 and decrementing at weight 4. Alternating up and down counters in this manner has the advantage that a carry emerging from an up counter *cancels* the incoming down count of its neighbor to the left, and a borrow cancels the neighboring up count.

REFERENCES

1. Meagher, R. E., and J. P. Nash. "The ORDVAC, Review of Electronic Digital Computers." *Proceedings of the Joint AIEE-IRE Computer Conference.* New York: American Institute of Electrical Engineers, 1952, pp. 37–43.
2. Wheeler, D. J. and J. E. Robertson. "Diagnostic Programs for the Illiac." *Proceedings of the IRE,* 41-10 (1953), pp. 1320–1325.
3. Williams, Charles R. "A Review of ORDVAC Operating Experience." *Proceedings of the Eastern Joint Computer Conference.* New York: Institute of Radio Engineers, 1953, pp. 91–95.
4. Robertson, J. E. Odd Binary Asynchronous Counters, *IRE Transactions on Electronic Computers,* EC-5, No. 1, (1956), pp. 12–15.

The ORDVAC

R. E. MEAGHER J. P. NASH

THE ORDVAC is a general purpose machine which has been built by the University of Illinois for the Ballistic Research Laboratoriesat Aberdeen, Maryland. The design has followed in a general way that of the computing machine discussed by Burks, Goldstine, and von Neumann in the "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" issued in 1946 by the Institute for Advanced Study.[1]

We are indebted to the Computer Project at the Institute for Advanced Study for much information and many suggestions such as the basic philosophy of an asynchronous machine with direct-coupled circuits. Furthermore some of the circuits in our machine are the same as those of that machine and some others are similar but differ in detail.

It is convenient to display the characteristics of the machine by means of Table I. Some of these will be described in more detail in the appropriate sections which follow. For this purpose we shall divide the machine into five parts: the arithmetic unit, the input-output, the memory, the control and the power supply. Two general views of the machine, with and without its covers, are shown in Figures 1 and 2.

Arithmetic Unit

The arithmetic unit is made up of three parts: the registers, the adder, and the digit resolver. The registers are four in number, the accumulator, the arithmetic register, the number register, and the order register. The register gates and flip-flops used in the ORDVAC follow exactly designs furnished by the Institute for Advanced Study. Of the registers named above, the latter two are nonshifting registers each·of which consists of 40 binary digits held in 40 flip-flops. Information is communicated through them to the arithmetic unit and control from the memory. The number register holds addend, multiplicand, and divisor for arithmetic operations. The order register receives from the memory all orders to be handled by the control.

The shifting registers (accumulator and arithmetic register) each consist of two rows of 40 flip-flops, one row above the other. Since a nonshifting register contains one row of 40 flip-flops, two nonshifting registers can be made out of one shifting register, and the appearance of the ORDVAC is that of a machine with three double registers.

In addition to the two rows of flip-flops, a shifting register also has two rows of gate tubes. These gate tubes furnish four gates for each stage of the register—two "up" gates and two "down" gates. The "up" gates transfer straight up, while the "down" gates transfer down one place to the left or one place to the right. The bottom row of a shifting register holds the digits while they are being sensed during an operation; the top row is used as transient storage. A shift is accomplished by clearing the upper row, gating up to it from the lower row, clearing the lower row, and finally gating down either one place to the left or one place to the right. When shifts are made in this way information is not cleared from one location until it has been deposited in another.

Figure 3 illustrates part of a register, the down gates into a lower flip-flop being shown. The plate supplies of a flip-flop come from busses which are normally at $^+150$ volts but which can be lowered for the purpose of clearing the flip-flop to either state. Thus if the supply to pin 1 is dropped, the flip-flop is put into the 1 state. The 1 state is arbitrarily defined as the state when pin 2 is high and is distinguished by a neon on pin 2.

The plate of a gate tube goes to the plate of the flip-flop to which information is being transferred; the gate grid is attached to the grid of the flip-flop from which information comes. When the gate cathode is lowered, the gate tube conducts if the grid is high and changes the second flip-flop if it is connected to the high plate.

This means that the second flip-flop must be in the proper state for receiving the gated information. Suppose a down right gate is wanted. First, F_3 is cleared to the 0 state by lowering pin 2, thus making pin 1 high. Then the cathode of G_R is lowered. If $F_1=0$, pin 6 is low, G_R does not conduct, and F_3 remains in the 0 state. It is thus the same as F_1. On the other hand, if $F_1=1$, pin 6 is high and G_R conducts. This causes pin 1 of F_3 to fall and turns F_3 to the 1 state. Therefore F_3 is the same as F_1.

The gating up in the registers is similar to the gating down except that only one "up" gate is ever used. This results in an extra set of gates which are not needed for shifting operations and are thus available for other use.

Access to the memory is in parallel from the accumulator and information from the input is supplied serially to the accumulator, utilizing the shifting properties of the register to put it in.

Similarly, the output is made serially from the arithmetic register, it being possible for the memory to communicate with this register.

The adder is a Kirchoff adder. It is basically the same as that in the Institute for Advanced Study machine, but it differs in detail, largely because of the difference in power supplies. The addend and augend digits control gates from two constant current sources which can allow current to flow through a summing resistor. The carry from the preceding stage controls the input voltage to the summing resistor. There are four possible voltages across the summing resistor, depending upon whether the sum at that particular stage is 0, 1, 2, or 3. If it is either of the latter two, a carry is produced for the next stage.

It may be seen in Figure 4 that the summing resistor is a 10,300-ohm resistor and the constant current sources are cathode followers supplying 4.85 milliamperes. The carry causes the input voltage to the 10,300-ohm resistor to

Table I. ORDVAC Characteristics

Machine type	Parallel, Asynchronous
Register capacity	40 binary digits
Memory capacity	1,024 words of 40 binary digits
Adder carry time	$9^1/_2$ microseconds
Allowed carry time	13 microseconds
Addition time	42 microseconds
Multiplication Time (all ones, positive multiplier)	1,000 microseconds
Multiplication time (all zeros, positive multiplier)	570 microseconds
Division time	1,000 microseconds
Memory period	24 microseconds
Time to load entire memory	38 minutes
Time to print contents of entire memory	38 minutes
Number of tubes	2,718
Machine d-c power	8.3 kw
Machine a-c power	8.8 kw
Total primary power (including power supplies and blowers)	35 kw
Kind of input	5-hole Teletype tape
Input system (space is 5 holes)	Sexadecimal
Output system	Sexadecimal Teletype
Number of digits assigned to an order	9
Number of digits assigned to memory address	10
Number of orders available	Greater than 50

R. E. MEAGHER and J. P. NASH are both with the University of Illinois, Urbana, Ill.

drop from 210 volts to 160 volts. The result is that the four voltages are in 50-volt steps, being 204, 154, 104, and 54 volts respectively. The time required to propagate a carry through the 40 stages of the adder is $9^1/_2$ microseconds.

The digit resolver is required to convert the adder voltages back to digital information. The addend and augend come into the adder from the accumulator and the number register, and the sum is returned to the accumulator after passing through the digit resolver. It is the function of the digit resolver to distinguish those sums having the digit 1 left in a stage from those having a zero. The former are the quantities 1 and 3 (having binary representation 01 and 11) while the latter are the quantities 0 and 2 (having binary representation 00 and 10). The digit resolver thus furnishes 0 volts for the accumulator if the adder output is 204 volts or 104 volts and furnishes -40 volts if the adder output is 154 volts or 54 volts.

Input-Output

The input-output equipment consists of modified Teletype units of the kind which were developed by the Bureau of Standards for the Institute for Advanced Study. It operates at standard Teletype speed, although the input has frequently been operated at about twice standard speed. The input is by means of standard 5-hole Teletype tape of which four holes are used to represent numbers in the sexadecimal or base-16 number system by means of a binary code. A 40-digit binary number is then written as a 10-digit sexadecimal number.

Output from the machine can be either to a tape punch or to a teletypewriter. In either case it is in the sexadecimal system, although if it is desired the output can be programmed and converted by means of a simple routine to decimal quantities.

The input and output are slow and in this facility the machine is unbalanced for some types of problems. At present consideration is being given to speeding both operatons up by a factor of about five, but beyond this no immediate increase in speed is contemplated.

Memory

The memory is of the electrostatic "Williams" type using 40 *3KP1* cathode-ray tubes.[2] Each cathode-ray tube stores 1,024 binary digits and can receive binary information from one digit of the accumulator and can release its stored information to the corresponding digit of the arithmetic, number, or order registers. Thus the total storage is 1,024 words of 40 binary digits. A slave cathode-ray tube and a 40-position switch are provided so that the contents of any one of the operating tubes can be viewed at one time. The cathode-ray tubes are operated with their deflection plates and second anodes at about 150 volts positive with respect to ground and with their cathodes at about 1,900 volts negative with respect to ground, giving an accelerating potential of about 2,050 volts.

Each cathode-ray tube is mounted in a horizontal position with its own shield consisting of a layer of copper, a layer of mu metal, and then a second layer of copper and a second layer of mu metal. Each cathode-ray tube is provided with an adjustment for intensity, focus, and astigmatism.

Associated with each memory tube is a chassis which can be seen in Figure 2 and which contains a 4-tube "video" amplifier with a gain of about 50,000 for a 1.5-microsecond pulse. The chassis also contains the logical circuit to provide for regeneration in the usual way and

Introductory Remarks

The following introductory remarks by **Dr. Herman Goldstein** *of the Institute for Advanced Study, preceded the presentation of this paper:*

Primarily, I wanted to say just a few words about the over-all history of the computer field with minor remarks about some of the things that characterize the machines that Dr. Meagher is going to discuss.

The history of the subject is extremely ancient. There has been and always will be a great need and desire not only in the obvious field, such as engineering, where answers are wanted for quite specific and quite clear reasons, but also in both pure and applied mathematics for results of an essentially computational nature. The desire to compute, then, is very old, but the mechanisms, as we well know, have been rather slow in development in any substantial way, and any of the things of the sort we are interested in can be traced to the confluence of two evolutionary streams in the last war. On the one hand, there was a development of electronic techniques in radar guidance and, on the other hand, an extremely pressing need among mathematicians for practical results. I think our present field of modern computation represents the confluence of these two evolutionary streams. But I would like to say a word about the particular thing which I think makes it possible.

Clearly, what one does in a computing machine is to represent mechanistically what a human computer performs. If it were not for the deep functions of the human intellect we would never be able to produce a machine. Fortunately, if one looks at what goes on when one does a computation, he finds it is possible to relate or resort to the ordinary vocabulary and transmit it to a computer in a dozen or so different words. Also there are problems of analysis characterized by this fact: that the total set of instructions in the "Pidgin English" of a dozen or so words, would run roughly to one printed page of instructions. One wants then to do this same page over and over again, perhaps each time changing some of the parameters which enter into the page. Therefore, it is possible to mechanize computational work and it is this fortunate fact, I think, which is the key to the whole success of the modern work from a logical level, at least.

Now, a word about the type of machines which Dr. Meagher is going to describe. It is true, there are a fair number either built or being built. All differ in certain respects but are of sufficient fundamental likeness that it is possible for me to say a word about them collectively, and for Dr. Meagher to essentially describe them all in describing the one at the University of Illinois.

The first thing about these machines I would like to mention is that they operate in a binary number system as distinguished from decimal machines. I do not particularly want to go in to processing methods and number bases, and of course the human does operate at base ten, but it is difficult to understand why there is any necessary reason for a mechanism to operate in the base ten. Therefore, one asks what is the best number of base to operate from? One thing that immediately strikes one—all machines have a logical and arithmetical part. The nature of mathematical logic is its binary character. So there is some reason for feeling the binary system has a certain advantage from that point of view. From the point of view of the arithmetic, the fact is that most of the units that one can produce electronically are fundamentally binary in character and to use another number base, such as base ten, one has to pyramid them. There is necessarily a certain wastefulness in such operation, because it takes four binary devices to produce a decimal digit. So one obtains, essentially, using groups of four binary devices, the ability to build up a decimal system. Since in this case it is possible to recognize 16 states and one uses only 10—six are lost. But I will not go into the subtleties that go into this machine. Suffice to say, it is binary. Secondarily, it is parallel, namely that all of the digits of two numbers are operated on simultaneously instead of starting seriatem with a pair and then a pair, et cetera.

The third classification is that these machines all use Williams' tubes; all use 40 of them in parallel, I believe. That is, all the deflection plates are swung in parallel with one digit for each of the 40 tubes. Some use 2-inch, some 3-inch, some 5-inch tubes.

I think this, by and large, describes the broad features of this type of machine.

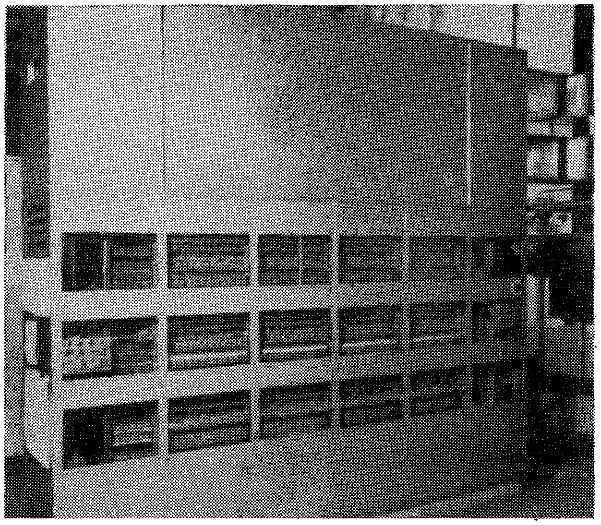


Figure 1. A front view of the ORDVAC showing the registers

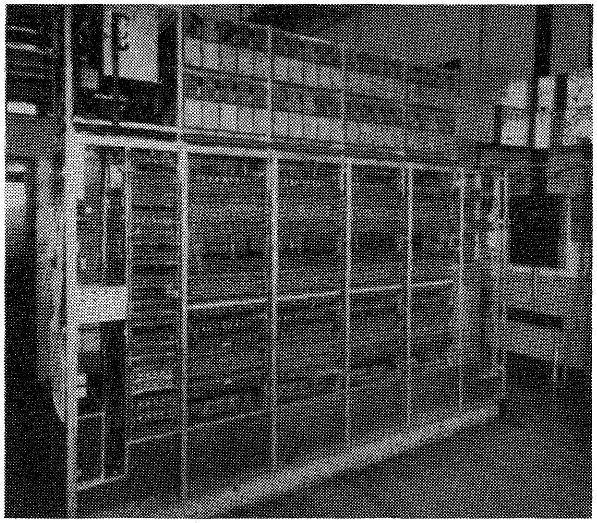


Figure 2. A front view of the ORDVAC without covers. The upper section contains the memory composed of 40 *3KP1* cathode-ray tubes and their associated chassis

for transfers of information to and from the appropriate registers. The output of this chassis is capacitively coupled and then d-c restored to the grid of the cathode-ray tube. The "2-dot" system of storage is used where the sensing position of the beam is called the dot and the other position is called the "dash." The beam is on for about 1.2 microseconds in the "dot" position and about 2.5 microseconds in the "dash" position. The period between cycles of memory operations has commonly been set at 24 microseconds, although all of the useful operations, including the clearing and gating of information, require about 16 microseconds.

The voltages for the deflection plates of the cathode-ray tubes which specify the position of the beam for each of the 1,024 memory addresses are obtained from an "address generator" which converts ten binary digits into 32 vertical positions and 32 horizontal positions by adding the currents from five binary stages for each. Some special care has been taken to insure that the deflection voltages are free from noise by providing special regulation for two of the voltages and by suitable limiting of the voltages on the wires which carry the ten binary digits to the "address generator."

It seems appropriate to summarize the useful performance of the memory with the following remarks.

It is necessary to check the adjustments of each memory tube daily by observing the video signals on an oscilloscope when the memory is in active use so that the signals corresponding to both dots and dashes can be seen. It is usually necessary to adjust about two or three intensities each time this is done.

We are currently using 40 *3KP1* tubes having a total of about 50 flaws on their phosphor screens which are bad enough to prevent satisfactory storage. About 15 of the 40 tubes have no flaws of this type. A process of tube selection is presently under way to reduce the flaws but the present 40 tubes represent the best 40 out of about 175 tubes. Currently we are operating by moving the raster on all 40 tubes in such a way that it does not use any of the flaw positions.

If one defines the "read-around-ratio" as the number of times that a dash can be read into or out of a single address without causing a nearby dot to change to a dash, and assuming that no regenerations occur, then it is possible to state that out of the 40,960 storage locations about three have a read-around-ratio as low as 20, about 20 have a read-around-ratio as low as 32 and about 200 have a read-around-ratio as low as 50. The way in which these results have been obtained is explained in a later section.

Control

As has been mentioned the control of the ORDVAC is direct coupled and asynchronous. Rather than have an external timing device for signalling each of the operations it is to perform, it operates by having each of its operations signal the one to follow. The speed with which it operates is therefore determined by its own ability to carry out the sequencing operations which are required. Since it is direct coupled the machine will simply stop if no signal is supplied for an operation. On the other hand, the machine can be made to stop by deliberately inhibiting a signal.

Let us consider one flip-flop F and two operations A and B as shown in Figure 5. When F is in the 0 state, A occurs and causes F to turn to 1. This initiates B which returns F to 0, and the sequence is repeated. Thus one flip-flop can be used to sequence a pair of operations. In the same way, n flip-flops can be used to sequence 2^n operations.

There are obvious defects in the simple example just cited. In the first place, the signal from A may turn F to 1 before A has had a chance to do its work. If this happens, the enabling signal to A will disappear and A will never be completed. Secondly, B may take place while A is still on. This could have disastrous results if the two signals were related, as, for example, when A clears a register and B gates information to that register. If B gates while A is still on and, worse still, if B never gets really turned on, the information would never be gated.

In order to prevent failures of these kinds, safety circuits have been designed into the ORDVAC control. These circuits do the following things:

1. They make the turnover requirements for control flip-flops more stringent than the turnover requirements for the flip-flops being controlled. Therefore, if the latter do not turn, neither will the former, and the machine will stop.

2. They require that when a control flip-flop is being turned over and sensed, the

signal from it be used only when it has been positively turned.

3. They require that before an operation in a sequence can occur the previous operation must not only have taken place but that it must also have been turned off.

Examples of circuits of these kinds may be found in the shift sequencing control. Shifting in the ORDVAC requires four operations, two clears and two gates, and these are sequenced with two flip-flops. Let us take a look at one of these flip-flops, the one which is turned over by a register gate bus. But first let us recall the register gating.

When the cathode of the gate tube is pulled down from +10 volts to −10 volts the contents of the second flip-flop will transfer if the grid was positive. Flip-flop grids are either 0 volts or negative about −20 volts. Since the cutoff of the *6J6* used for gating is about −4 volts, the gate will start to conduct when the cathode is at about +4 volts, and the transfer will have occurred by the time the cathode gets to 0 volts.

The same gate bus which pulled down the gate cathodes in the register is used to turn the control flip-flop F (Figure 6). But now the gate tube G is pegged at −5 volts instead of at 0 volts. This gives a 5-volt safety margin in the turnover of F, and it can be safely assumed that the register flip-flops turned over it F did.

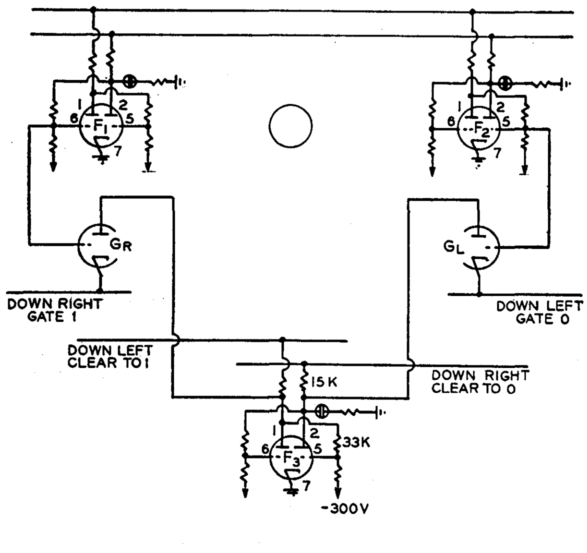
Negative signals are used in nearly all of the ORDVAC control. Before F was turned, pin 6 was negative. After it is turned by G, pin 5 is negative, and one might be tempted to take the signal for the next operation from here. But this would be a mistake, for pin 5 might go far enough negative to initiate the next operation before F is safely and permanently turned over. Therefore the signal will be taken from the positive-going grid 6 which is the last moving element of F. It will be sensed with the inverter N_1 which is built just like F. Because of the cutoff properties of N_1 it will not conduct until the grid gets up to −4 volts, and by then we know that F is safely over. The output of N_1 therefore says that the register gate bus has gone down and that F has been turned over. The turnover of F will shut off the gates in the register.

We still need to know that the gate bus is back up again before proceeding with the next step. This information is obtained from the inverter N_2. With pin 5 pegged at +5 volts, the signal from N_2 will not be negative until the gate bus is safely off again. The outputs of N_1 and N_2 then go to the "and" circuit A which starts the next operation when its cathode goes negative.

This is the kind of philosophy which has prevailed in the design of the control. Throughout the control an effort has been made to assure the safe operation of each step. If any step should fail, the result is not a loss of the information in the register but merely a "hanging up" of the machine. The control waits until the proper enabling signal comes along, and then it proceeds. This kind of construction makes it possible to test the operation step by step by putting switches in appropriate places so that the turnover of flip-flops can be controlled. One such place is in the cathode of the gate turning over the flip-flop F in the previous example. If this switch is open, the register gate will not turn off and the machine will wait until the switch is closed.

A price must be paid for these safety features, and the price, of course, is speed. The machine will run faster if it is not necessary to wait for checks on the operation. It will also run faster if operations are done as much as possible in parallel. Here again, although the ORDVAC is a parallel machine, a price has been paid in speed for convenience in doing certain control operations sequentially rather than in parallel.

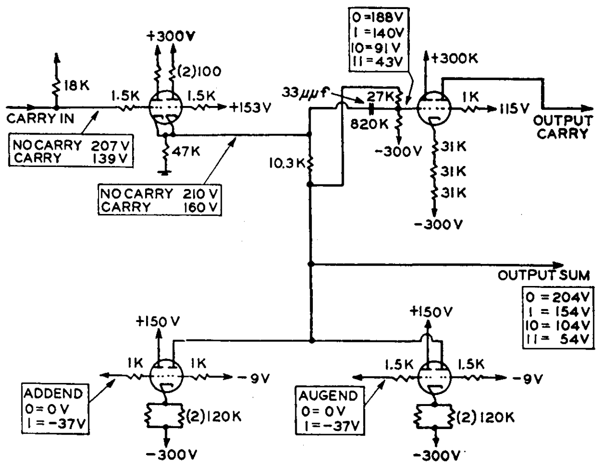
The ORDVAC memory is synchronous with a period of 24 microseconds. When the control is carrying out operations which do not involve the memory, it works independently while the memory regenerates its storage locations. But if an operation requires use of the memory, the control and memory must be synchronized for one 24-microsecond interval, called an action cycle. In order to get into synchronization the control furnishes a signal to the memory. Since this signal may occur at any time during the 24-microsecond cycle, the control must then wait for an appropriate memory pulse, the action sense pulse, to come along. When this pulse comes, the action cycle normally takes place. During this cycle information is transferred to or from the memory, the necessary clearing and gating operations for these transfers being executed more or less directly



FLIPFLOP STATE	PIN 1	PIN 2	PIN 5	PIN 6
0	HIGH	LOW	HIGH	LOW
1	LOW	HIGH	LOW	HIGH

Figure 3 (left). Register circuit showing two upper flip-flops, one lower flip-flop and the "down" gating system

Figure 4 (below). Analogue adder showing input for addend and augend, output, and the carry circuit



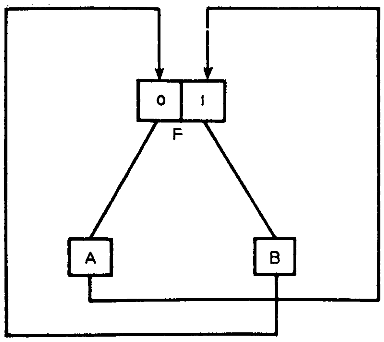


Figure 5. A single flip-flop F used to control two operations A and B in sequence

by the memory pulses themselves. This one case where the memory clears and gates in the registers is the only synchronous control operation in the machine.

The action cycle ends when the next action sense pulse occurs 24 microseconds after the one which began the cycle. At this time a "have used memory" signal goes to the control to indicate that the use of the memory has been completed. The memory returns to regenerating, and the control resumes its asynchronous activity.

The synchronization process requires two flip-flops. One distinguishes between an action cycle and a regeneration cycle. The second distinguishes between the time before the action cycle, when the memory is still regenerating, and the time afterward when it has returned to regenerating but while the "have used memory" signal is on.

A large part of the control is devoted to setting up the proper clear and gate sequences for the registers. Fundamentally, of course, everything that is done in the arithmetic unit must be a combination of sensing flip-flops, adding, clearing, and gating. The decisions about these things are made after certain combinations of digits, called orders, have been decoded with logical circuits. The number of orders that is actually necessary for the convenient coding of the kinds of problems which the ORDVAC is expected to solve is probably between 20 and 30, and this many orders can be described with 15 flip-flops, ten of them being needed for memory addresses. A decoding matrix using five flip-flops would then be used to cause one of 32 wires to be actuated whenever the corresponding order is presented. However (as suggested by the group at the Institute for Advanced Study) since there are 20 digits available for an order of which only ten are needed to describe the 1,024-memory locations, we can use as many as ten for an instruction. Actually nine have been used with the result that the number of tubes required for decoding has been decreased and the number of orders has been increased. The number of orders which has actually been used in programming is about 50, some of which were not forseen when the control was designed. A complete matrix is not used, the decoding circuits being made up of a number of submatrices.

The arithmetic of the machine is one which handles numbers in the range from -1 to $+1$ (actually excluding $+1$) and which carries negative numbers as complements relative to 2. This is the kind of arithmetic which was described by Burks, Goldstine, and von Neumann.[1] However, the methods described by them have not been followed in the arithmetic operations of this machine. In particular, the method by which multiplication is carried out is not used, as far as we know, in any other machine. (This method was suggested by Mr. J. E. Robertson of the University of Illinois.)

Multiplication without roundoff by two positive numbers poses no problems. If an operand x is negative, the machine holds the number $2+x$, and if the sign digit of x is ignored multiplication by a positive y gives the result $xy+y$ so that subtraction of y will give the desired answer. If x is the multiplier, there is no difficulty because y is available at the end and can be subtracted. But if x is the multiplicand, y is lost during the process of shifting to inspect its digits. Hence it must be subtracted during the stepwise formation of the partial products. The method proposed by Burks, Goldstine, and von Neumann which does this, requires varying the gating of digits to the adder as well as making a "correction" at the end.

Figure 6. Safer way by which flip-flop F can control an operation A

(A) Block diagram (B) circuit

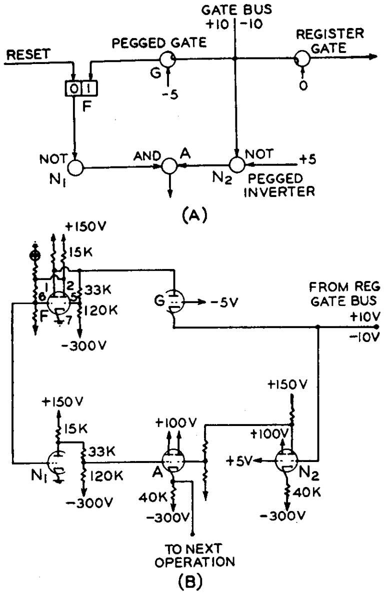


Table II. Multiplication with Negative Multiplicand Showing System Given in Burks, Goldstine, and von Neumann and System in ORDVAC

Multiplicand 1.101 Multiplier 0.101

Operation	Multiplier Digit	Accumulated Product
Burks, Goldstine, von Neumann		
(1) Add multiplicand without sign	1	0.101
(2) Shift right		0.0101
(3) Change sign and shift right	0	0.10101
(4) Add multiplicand without sign	1	1.01001
(5) Shift right		0.101001
(6) Add correction term 1.001		1 110001
ORDVAC		
(1) Add multiplicand	1	1.101
(2) Divide by 2		1.1101
(3) Divide by 2	0	1.11101
(4) Add multiplicand	1	1.10001
(5) Divide by 2		1.110001

The ORDVAC multiplication scheme makes no distinction between positive and negative multiplicands. This is accomplished by making an algebraically correct division by two at each step, and always adding the complete multiplicand with its sign if an addition is required. No corrections are required at the end. The same circuits are used as those which carry out the right shift order, this order giving a correct division by 2 in the accumulator. A sample problem may be followed in Table II.

The problem posed by the roundoff, which requires that the product be rounded to a sign and 39 places after the addition of 2^{-40}, has been solved by doing the roundoff before the multiplication begins. If there is no roundoff, the accumulator is cleared and the product xy is formed. If there is to be a roundoff, the accumulator is cleared, 2^{-1} is gated into it, and the quantity $xy+2^{-40}$ is formed. Since the accumulator holds only the sign and first 39 digits of the product, it holds the rounded product.

This method of rounding off has the advantage of being fast, because there is no need to do an additional step at the end which involves waiting for carries in the adder, and it also provides the facility for gating the digit 2^{-1}, into the accumulator for other orders. In connection with a shift order a digit can be put any place in the accumulator without requiring additional memory storage space.

The division process is different from the nonrestoring process described by Burks, Goldstine, and von Neumann and is almost the inverse of the multiplication process.

Power Supplies

The machine uses about 8.8 kw of a-c power to operate the filaments of the vacuum tubes.

The d-c power consumption is:

−2,000 volts	0.09 ampere
− 300 volts	16.0 amperes
+ 100 volts	10.1 amperes
+ 150 volts	3.8 amperes
+ 300 volts	5.2 amperes
+ 680 volts	0.4 ampere

The −2,000-volt and the +680-volt potentials are obtained from vacuum tube regulated power supplies which were built at the University of Illinois.

The intermediate four voltages supply the bulk of the d-c power for the machine and insofar as possible all circuits have been designed to use these voltages directly without additional regulation or "bleeders." In a few cases such as adder circuits "odd" voltages are required and these are furnished by a "bleeder" which supplies voltages for vacuum tube grids where the current drain is very low. These four "main" voltages are furnished by commercially built and controlled rectifier units which are regulated against line voltage changes and fast and slow load changes to ±2 per cent.

All of the d-c loads are divided into separate circuits of about 3 amperes each and arranged with individual fuses in such a way that the failure of any individual circuit will open a relay thus turning off a "holding circuit" on all of the d-c power.

Operation of the ORDVAC

To date no mathematical problems other than trivial ones have been solved by the ORDVAC. A considerable number of test routines have been run, some of them having 100 words or more and using some 25 of the available orders of the machine. Most of the more complicated routines have been concerned with testing the memory for read-around-ratio, although they serve also, of course, to test the arithmetic unit and control at the same time.

One of the more useful routines is one which tests every storage location of the memory for its read-around characteristics. The raster now is arranged so that the spots form hexagons. The routine is put into the top of the memory and causes the first spot in the lower half to be bombarded a given number of times with dashes. It then inspects the surrounding spots, which have been previously cleared to dots, to see if any have become dashes. If so, it causes the tube number (0 to 39) and address of the bombarded spot to be printed. This is repeated for all of the addresses in the lower half of the memory. The routine then transfers itself to the lower half and inspects the upper half in the same way. At the end of the complete inspection it changes the number of bombardments and repeats the process.

A program for the over-all testing of the machine has been run. This program generates a set of 352 pseudo-random numbers b_i and stores them in successive memory locations. It then performs multiplication and divisions of b_i by b_{i+1}, checking multiplication results by multiplying in both directions and comparing and checking division results by multiplication. If all of these are correct, the numbers are transferred to 352 other locations and the transfers are checked. If there is no failure, i is increased by one and the process is repeated, transferring each b_i to a different address this time. When i reaches 352 (a major cycle), the process starts again.

At any failure the machine prints pertinent data and stops. Upon being started again it goes through a subroutine which checks all of the principal orders, trying to find the cause of the first failure. If an order fails, the machine again furnishes information and stops.

This program ran continuously for 12 hours and then failed when one memory location changed from a dot to a dash.

The machine has been "on" about 2,900 hours and during this time about 190 tube failures have occurred. However, it should be remembered that the machine has "grown" during this time and only about 1,000 of 2,700 tubes have actually been in service during the full interval.

The cathode-ray tube life is inadequately noted by our system because of the selection process which has been in progress.

Conclusion

The work on the ORDVAC started in the spring of 1949 and has been supported by a contract from the Ordnance Department for $250,000. The University of Illinois has provided a comparable sum which is to lead to a second machine. The Ordnance contract was concluded on October 31, 1951 and since that time the ORDVAC has been under test. It is currently expected that it will be moved to the Ballistic Research Laboratories at Aberdeen about February 1952. Its presence at the University until then allows a further understanding of its operation and use, and aids in the work on the machine for the University of Illinois.

References

1. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, **A. W. Burks, H. H. Goldstine, John von Neumann.** Institute for Advanced Study (Princeton, N. J.) June 1946.

2. A Storage System for Use with Binary Digital Computing Machines, **F. C. Williams, T. Kilburn.** *Proceedings*, Institution of Electrical Engineers (London, England), volume 96, part II, number 50, April 1949, pages 183–200.

Discussion

Charles Corderman (MIT): First, speaking of the tubes you have in the ORDVAC, what is the consistently obtainable read-around ratio, and of the tubes that do not show flaws, what is the average read-around ratio?

R. E. Meagher: We usually do not check the average read-around ratio because the lowest read-around ratio is the one which would limit the use of the machine. Testing any one address may give read-around ratios of several hundred, but we have to impose an operating limit of something like 16, at this time.

David Mayer (Philco Corporation): I would like to ask for a definition of "read-around ratio". My second question is, are you using crystals in the machine? Third, how do you perform your division and subtraction operations?

R. E. Meagher: If the read-around ratio is "n", it means that we can write a dash or read a dash "n" times at that particular address without causing an error at any nearby address or any other addresses. That means when I say the read-around ratio is 16, there is some address on some tube which will fail when you hit it 17 times. We have no crystal diodes at all in the machine.

Division is carried out in the machine by a process which may be thought of as being the inverse of the multiplication process which you saw. If we confine our attention to the division of one positive number by another, then at each step of the process we form a partial remainder which is held in the accumulator. At the beginning we have the dividend in the accumulator. We subtract the divisor from the partial remainder to form a new partial remainder. If the sign of the difference is positive, the new partial remainder is the difference shifted left one place and the quotient digit is a 1. If the sign is negative, the new partial remainder is the old partial remainder shifted left one place, and the quotient digit is a 0. The quotient is shifted with the accumulator, and digits are inserted at the right.

A Review of ORDVAC Operating Experience

CHARLES R. WILLIAMS†

Introduction

THE ORDVAC is one of three large-scale electronic computers located at the Computing Laboratory of the Ballistic Research Laboratories at Aberdeen Proving Ground, Maryland. It is the newest computer at the laboratory having been delivered in March, 1952. It operates in the binary number system in a parallel asynchronous manner, and it uses an electrostatic memory. Input to the machine is by punched teletype tape or punched IBM cards. Output from the machine is obtained on punched IBM cards, a teletype page printer, or punched teletype tape.

During the period from March 10, 1952 to October 2, 1953, the ORDVAC was available for computation a total of 8,410.18 hours. The total engineering time was 4,189.17 hours, and the remaining 1,104.65 hours was standby time. Standby time includes all the time during which no attempt is made to operate or service the machine. In one week the standby time is 168 hours minus the sum of the total engineering time and the total available time.

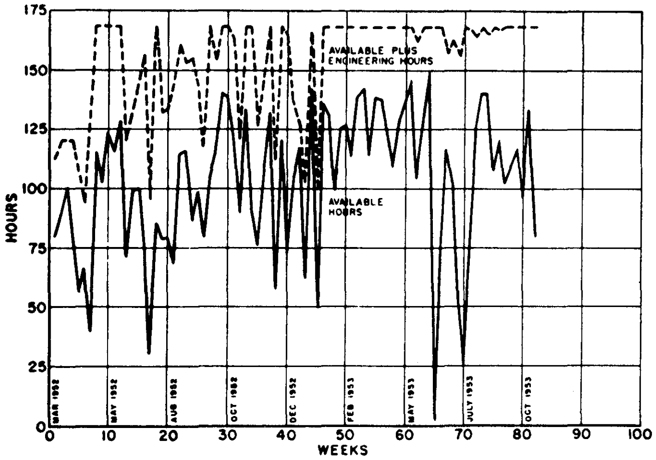


Fig. 1—ORDVAC weekly performance chart.

Machine Performance

The ORDVAC's performance to date is best described by the curves of Fig. 1. The lower curve is a weekly plot of the number of hours the ORDVAC had been available for immediate use. The period described is from March 10, 1952, when the ORDVAC was accepted by the laboratory, until October 1, 1953, a period of 82 weeks. The upper curve of Fig. 1 represents the number of hours each week that the machine had power applied to it. The distance between the upper curve and the 168-hour line thus represents standby time, and the number of hours between the curves represents engineering time.

During the earlier periods there was a considerable amount of standby time. The lack of sufficiently trained personnel and the lack of programs did not make it desirable to maintain the machine over the weekends. As more programs were completed, however, and the machine load increased, it was soon learned that operating the ORDVAC without interruption reduced the number of troubles encountered on Monday morning and enabled it to be made available for use at an earlier time. At present, the only time that ac power is removed from the computer is for repair of its cooling equipment, or because of a building power failure. Standby time has averaged 13.54 hours per week.

Fig. 2 illustrates the use that has been made of the ORDVAC by a comparison of the various classifications to which machine time is charged.

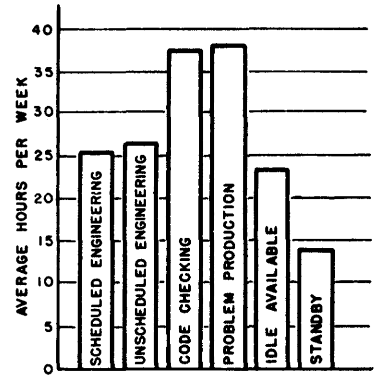


Fig. 2—Machine-time classification chart for an average week.

Within the period under discussion at present, an average of 37.17 hours per week was spent on code checking, 38.29 hours per week spent in production, and 23.03 hours per week classified as idle time. The title "idle time" is not intended to indicate that the machine is not in use during that time. The machine is made to run continually a test called Leap III, but it is available for other use at any time. Should the Leap III routine fail, the following time is immediately classified as unscheduled engineering until the cause of the error has been corrected, and the routine has successfully operated for fifteen minutes.

Recently, after all scheduled problems have been run, the remaining time has been used for problem duplication in an attempt to discover any errors that may have occurred during a regular production run. Normally

† Computing Laboratory, Aberdeen Proving Ground, Md.

problems are not duplicated unless suspected of being in error or, as just stated, unless time permits. Sufficient records have not been kept, but investigation has revealed that approximately 10 hours per week are lost as a result of errors that have occured. It should not be assumed, however, that the number of errors vary directly with the number of hours lost, because an error may not be detected until several minutes or even hours after it has occurred.

Perhaps one of the best yardsticks for measuring error frequency is the number of errors that occur over a given period while the machine is idle and running the Leap III test. During a three-month period in which the test was run 546.75 hours, the Leap III test failed 141 times due to the memory and 31 times due to the arithmetic unit. The average error-free running time was 3.2 hours. Again this figure is somewhat misleading in that the daily error frequency may vary widely. The time between the errors indicated above varied from a few minutes to a maximum of 24 hours.

Daily Schedule

The normal operating schedule includes three periods each day during which the performance of the ORDVAC is checked. The period between 0800 and 1000 hours is set aside for engineering purposes; at this time small changes and improvements may be made, troubles that may have developed during the night shifts are cleared, and, finally, the machine is tested by various self-checking programs. Tests are again made at the conclusion of each working day and at midnight. Upon the successful conclusion of the tests, the ORDVAC is released to the mathematicians who use it during the day. On the night shifts most of the production work is carried out by two technicians per shift who are in charge of the machine. They are not expected to understand the programs that are run, but they follow explicit instructions that are left by the mathematicians. This system has worked quite well. Usually, when a coder arrives at the laboratory in the morning he can expect to find that his program has been completed.

In the course of an average week 30 different problems are placed on the ORDVAC. Each problem may be placed on the machine several times each week. Ballistic computations, such as firing and bombing tables, and trajectories for guided missiles and rockets, form about 40 per cent of the work done by the ORDVAC. Vulnerability computations and data reductions form 25 and 20 per cent respectively. Research problems form 10 per cent and systems tests form 5 per cent of the work.

Testing the Machine

Three classes of tests are used to test the units of the ORDVAC. A read-around-ratio test is used for checking the memory; an input-output test is used to test the teletype and card-handling equipment; and the Leap III test already mentioned is used to test the arithmetic and control section and to give a further check on the memory.

Any electrostatic-type memory is subject to a fault which is known as read-around-ratio. In this paper, read-around-ratio refers to the maximum number of times that any one position in the memory may be consulted without causing errors in adjacent positions. A routine is used to scan the entire memory to discover any position or area that may be in an unnecessarily poor condition.

The present memory uses a three-dot system, which has the characteristic of being susceptible to failures of both ones changing to zeros and zeros changing to ones, should the read-around-ratio exceed a safe figure. Two tests are used to determine the read-around-ratio. They are similar except for the method of sensing a failure.

The routine is placed in the lower part of the memory from which it automatically proceeds to test the upper half. A number read into the machine from teletype tape determines the number of times each spot is bombarded before the adjacent spots (which had been previously cleared to ones or zeros, depending on the test) are scanned for failures. After half of the memory has been tested, the program is transferred to the tested area from which it proceeds to test the second half. Failures are indicated by a printed word identifying the cathode-ray tube involved and the address that was consulted. Up to a point, failures can usually be eliminated by the proper adjustment of the voltages on the cr tubes.

The input-output units are tested by a simple process of reading identical information into the machine from both teletype tape and cards and immediately punching out the same words. Comparison is made automatically, and discrepancies are printed out on the teletype page printer.

Perhaps the most thorough test is the Leap III test mentioned earlier. It is a revised version of the original Leap frog test written for the ORDVAC while it was still at the University of Illinois[1]; its name is derived from the manner in which it moves itself through the memory. In a period of about fifteen minutes, the routine moves in such a way that each order in it has occupied every memory address. Throughout the leaping process a self-checking system of arithmetic operations using pseudo-random numbers is carried out. If an arithmetic or storage error is detected, a set of twelve numbers are printed to reveal the nature of the error.

Occasionally the routine may fail in such a way that no useful information is obtained from the printed matter if, indeed, anything is printed at all. In this case a special routine is used to search the entire memory for the location of the program and then compare it with a correct copy.

It can be seen that the ORDVAC is used as much as possible to test itself by a highly repetitive use of its

[1] For a more detailed discussion of a similar test, see D. J. Wheeler and J. E. Robertson, "Diagnostic programs for the ILLIAC," Proc. I.R.E., vol. 41, pp. 1320–1325; October, 1953.

components. In this way, marginal and intermittent errors may be detected that would otherwise not be found. The usual memory errors are not to difficult to repair; the causes of arithmetic errors are somewhat harder to detect as their effects may propagate. Intermittent failures are especially troublesome if the error frequency is low. In this case an attempt is made to increase the error rate by using special routines which strain the circuits to their limits, by vibrating the circuit components, by varying filament or supply voltages, or by a combination of the above.

Engineering

It was mentioned before that during the nineteen month period under discussion the total amount of engineering work on the ORDVAC amounted to 51.36 hours per week. Broken down into scheduled and unscheduled engineering columns, the time was about evenly divided. They were 25.27 and 26.09 hours per week, respectively. Any engineering which necessarily interrupts the normal operating schedule without advance arrangement is classified as unscheduled engineering. In addition to the time taken each day to test the machine, scheduled engineering includes all the time which is taken for modifications to the machine. The major effort which has been made in the way of modifications to the ORDVAC has been devoted to the input-output system and to the memory.

Input-Output Modifications

The ORDVAC was delivered to the Ballistic Research Laboratories having as input a standard speed, five-hole teletype tape reader, and as output, a teletype page printer. Using this equipment, the time necessary to load the entire memory of 1,024 addresses was 38 minutes. The time necessary to print the contents of the entire memory was the same. No change has as yet been made in the page printer. It was realized quickly, however, that a change was necessary in the input in order to speed up the operation of the computer. Within a few weeks after the ORDVAC was placed in operation a control circuit had been designed and built for a modified tape reader which now allows information to be read into the machine at five times the previous rate; that is, the memory can now be filled in 7.5 minutes. This unit has performed exceptionally well, but input and output are still the big bottlenecks to more efficient use of the ORDVAC.

It is interesting to note the manner in which the University of Illinois has in the past used the ORDVAC remotely. The University would send to the Ballistic Research Laboratories a program through regular teletype channels which would be placed in the ORDVAC after insuring that the program had been received correctly. Answers were obtained on tape and returned to the University by mail or by the teletype channels. Thus the ORDVAC is available to any laboraory in the country having the necessary coding staff.

The most important change in input-output has been the addition of card-handling equipment. Before March, 1952, it was realized that card-handling equipment was desirable. The ORDVAC was to be placed in a laboratory with two other large-scale computers, the ENIAC and the EDVAC. The ENIAC already used punched cards, and such a system was being devised for the EDVAC.

There were two possible systems to be considered. The first was an external system in which information punched on cards was to be automatically converted to its binary equivalent by external apparatus. The second was an internal conversion system based on utilizing the ORDVAC itself to accomplish the necessary transformation. The second method was chosen.

The advantage of the internal system lay obviously in the elimination of the necessity for extensive additional equipment; and since established machine operations were utilized, the maximum in reliability was obtained. Another advantage was that less time was required to complete the addition of the equipment to the ORDVAC. The greatest disadvantage of the internal system was that a significant portion of the memory capacity was committed to the conversion program. The space required for both input and output conversions has amounted to 200 words, approximately one-fifth of the memory. Recent improvements of the memory, however, have made it possible to reduce the size of the program.

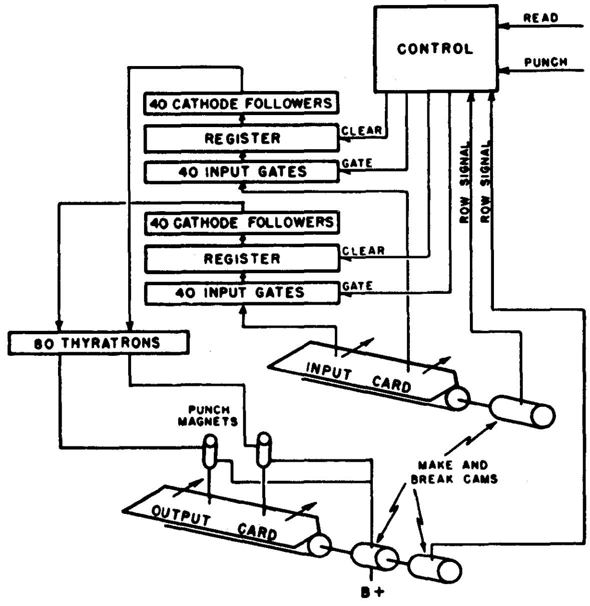


Fig. 3—Block diagram of the card input-output system.

The system is based on double-register gating in which the entire 80-column output can be gated to or from the ORDVAC simultaneously. A block diagram of the system is shown in Fig. 3. An electronic control circuit which is actuated by signals generated in the card punch or card reader allows the contents of the ORDVAC registers to be stored in the memory or changed in the interval between punching or reading successive rows on a card.

This system has proved quite reliable and is preferred to tape by the mathematicians. The usual practice is first to prepare a program on tape and immediately transcribe it to cards with a special routine that automatically reads from the tape and punches, in binary form, twenty-four words per card. Cards are used thereafter.

Several forms of input may be used. One form, just mentioned, in which 24 words per card are used, allows the memory to be filled in about 30 seconds. Another form of input, used for decimal data input and for which the 200-word input-output conversion routine is required, allows only eight decimal words to be punched on a card. Each card may be read into the ORDVAC in 1,033 milliseconds; this is equal to a rate of about 60 cards per minute. Data cards of this type may be used on either the ORDVAC or the ENIAC and may be handled with the usual card-handling equipment. By a third input method, 12 words are read from a card and stored in the machine by a repetition of an "order pair."

Memory

The electrostatic memory has always been the ORDVAC's weakest unit. Prior to June, 1953, the memory operated rather consistently at a read-around-ratio of 10 to 16. A restriction of 10 placed on the coders resulted in slower routines or routines that occupied a larger portion of the memory. In addition to read-around-ratio difficulties, there existed the problem of obtaining cr tubes whose screens were free of impurities. Type 3KP1 cr tubes were used for storage. Approximately 25 per cent of those tested were acceptable; most of the rejections were due to impurities which could cause improper storage. Until May, 1953, approximately 60 cr tubes were removed from the machine for various reasons: About two-thirds of the total number were removed because of the impurities in their screens; five were removed without proper cause; and the remaining tubes were removed because voltage adjustments could no longer hold the read-around-ratio of those tubes above 10. No cr tubes have burned out under normal operating conditions, although five were burned out when subjected to abnormal conditions.

In May, 1953, it was learned that the University of Illinois was obtaining improved read-around by using a three-dot system.[2] In this system, as used in the ORDVAC, two dots represent a one, and three dots represent a zero. The first and third dots occur at the same location.

The change over from a two-dot system was made about June 1, 1953. Considerable difficulty was encountered with the sensing pulse; and, in order to strengthen it sufficiently so that its excursion would not vary with the number of zeros sensed, it was necessary to incorporate a pulse transformer into the circuit. The new system caused the read-around-ratio to be increased by a factor of three so that it then varied between 32 and 48.

In the hope of further improving the reliability of the electrostatic memory, a number of type C73376B cr tubes, which are under development by RCA, were obtained and installed in the ORDVAC. At this writing they have been in the ORDVAC 1,500 hours and seem to be operating satisfactorily. The read-around-ratio is at present guaranteed to be 80 and is usually 100; the flaw problem no longer exists since these tubes are virtually flaw free. Although the reliability of the memory has been increased, the improvement is not very noticeable in Fig. 1. Unfortunately, a series of arithmetic errors, which were for the most part due to a group of bad solder connections, temporarily counteracted the improved memory performance.

The greatest difficulty experienced with the new tubes was due to their centering characteristics. Upon installation a large number of tubes had to be immediately removed, because corners of the 1024 spot raster projected beyond the useful surface of the screen. Ten tubes were removed for this reason. The ORDVAC, like most machines using electrostatic memories, operates its cr tubes in a parallel manner; thus a positioning of the raster which is beneficial to one tube might render the other tubes completely useless.

A test was made on the centering of 40 type C73376B cr tubes and, for comparison purposes, on 40 type 3KP1 tubes chosen at random. The results indicated that the experimental tubes were somewhat inferior in this respect.

In the design of an electrostatic memory, careful consideration should be given to the problem of centering the raster in the cr tubes if the maximum benefit is to be obtained from the unit. It is believed that in a three-inch tube the undeflected beam should be positioned by some method internal or external to the tube to within 2.5 mm of the center of the tube face.

Three tubes have since been removed from the ORDVAC because of the deterioration of the quality of the signals presented to the regeneration amplifier. The signals degenerated to the point that adjustments of the tube voltages would no longer make the signals reliable.

Preventative Maintenance

In September, 1952, a tube-removal program was initiated as a part of a preventative maintenance program. Since almost all of the troubles encountered were the result of shorted or low emission tubes, it was believed that such a program would be helpful in eliminating a potential source of trouble. Blocks of tubes were removed at two-week intervals, and were replaced by new ones. The location and the number of tubes involved were dependent upon the area in which troubles had been occuring most frequently. The number usually varied from 50 to 100. At present the ORDVAC has operated about 18,000 hours, and virtually all of its 3,000 tubes have been replaced at least once. The effect of the

[2] J. M. Wier, "Recent Improvement of ILLIAC Memory," Univ. of Illinois Graduate College, Digital Computer Laboratory, Internal Report no. 45; March 25, 1953.

block-tube change was felt almost immediately, and the procedure is believed to be an important factor in the gradual increase of available time noted in Fig. 1.

Within a period of eight months, 850 tubes were replaced in blocks; out of these tubes, 394 would not pass the inspection given all tubes that are used in the ORDVAC. The tubes used in the computer are, for the most part, types 6J6, 2C51, and 5687. Upon inspection it was found that about 50 per cent of the 6J6 type were bad. The most frequent cause of failure in this type tube was "shorted" elements. "Shorted" is meant to include those tubes whose elements were joined by a high-resistance path so that a leakage current was detected upon inspection. About 30 per cent of the 2C51 type tubes were found to be bad. Again, the most frequent cause of failure was "shorted" elements. The greatest percentage of bad tubes was found among the 5687 type. About 75 per cent of these would not pass inspection. The majority of the rejections in this case was due to low emission and cathode-to-heater leakage. It is interesting to note that the ORDVAC had been operating rather satisfactorily with such poor tubes in use.

The major cause of trouble in the ORDVAC has been tube failures. The conservative design, in which a safety factor of two was used in the rating of components, has made other failures practically nonexistent. Occasionally, however, bad solder connections do appear, as noted before, and they usually appear in groups. Recently, the cause for a great many adder failures was eliminated by the discovery of a wiring error which caused 200 volts to be applied between the heater and cathode of 50 tubes. Such wiring errors are difficult to eliminate except through continued use of the machine.

Perhaps one of the greatest potential sources of trouble in a machine is dust or dirt. In almost any installation some dirt is certain to enter the cooling system regardless of the elaborateness of the filtering system. Simply exposing the components for necessary maintenance will allow a great deal of dust to enter the system. A computer using an electrostatic memory is especially susceptible to dust, unless it is elaborately protected, because the high-voltage wiring forms a fine precipitator. This has been one of the great sources of trouble in the ORDVAC—not great in the sense of occurrence, but great in the sense of damage that may be done.

During the summer of 1952, sufficient dust had collected on the wiring so that the 2,000 volts arced to ground in a number of places. The damage included several clamping tubes that were exploded, at least six-memory chassis that had to be replaced, and five cr tubes that were burned out. The arcing was eliminated by floating the high-voltage system and supplying it with a high impedance variable high voltage. In a darkened room, the arcs were both audible and visible, and they were eliminated by cleaning and separation of the wiring where possible. The manner in which the ORDVAC is constructed makes proper cleaning almost impossible, but a little care has thus far prevented a reoccurrence of the trouble.

Conclusion

Recent improvements to the electrostatic memory have removed certain restrictions on coders and have led to shorter and faster programs.

A preventative maintenance program consisting chiefly of periodic tests and a systematic exchange of tubes, along with the improvements to the ORDVAC, have resulted in an average of 103.10 hours out of a possible 154.46 hours each week over a period of 19 months being made available for computation. It is expected that even a higher number of available hours will be obtained in the future when procedures are perfected and operators gain more experience.

Discussion

Harvey Rosenberg (Burroughs Adding Maching Corporation): How many control relays are used? What was their reliability?

Mr. Williams: In addition to the few relays used to control the power to the ORDVAC there are six telephone-type relays and one stepping relay which are used in conjunction with the teletype output system. No trouble has been experienced with the power control relays; the stepping relay has required minor adjustment only three times in two years; and it has been necessary to replace two of the telephone-type relays because of broken contact arms.

B. B. Paine (Massachusetts Institute of Technology): Could you elaborate on the tube acceptance test program? Is preburning used?

Mr. Williams: Before all tubes are placed in the ORDVAC they are given four tests. The first is a very simple short test in which the tube is tapped manually while a neon is watched. No attempt is made to actually measure the resistance between elements. The second test is an emission test. The third test is a cut-off test. If I remember correctly, in this test about 20 volts bias is placed on the tubes and no more than about 50 microamperes plate current is allowed to flow. The fourth test is a filament-to-cathode leakage test in which no more than 50 ma current at 150 volts is allowed. Tubes are not pre-burned at present, and few tube failures can be credited to this fact.

E. L. Harder (Westinghouse Electric Corp.): Were errors detected in the teletype transmission from the University of Illinois? How many? How detected? How corrected? How was transmission of errors verified?

Mr. Williams: In answer to these questions I will go into more detail on the manner in which the tapes were handled. The University of Illinois initiated contact by transmitting their program directly to the Signal Corps office at Aberdeen Proving Ground, preceeding the program with instructions to the ORDVAC operator. Immediately upon receipt of the tape, it was transmitted back to the University of Illinois where it was compared with the original copy. If an error was detected upon comparison, the procedure was repeated until a correct transmission was verified. In a one month period during the summer of 1952 transmitting time averaged about forty-five minutes a day. There were several human errors made in this period, but there were only two mechanical errors.

Diagnostic Programs for the Illiac*

DAVID J. WHEELER†, AND JAMES E. ROBERTSON†, ASSOCIATE, IRE

Summary—**The diagnostic programs used for maintenance of the ILLIAC, the University of Illinois' digital computer, are described. The uses of diagnostic programs for fault detection, fault isolation, and periodic computer servicing are discussed. The characteristics of the "leapfrog" program, both as a detection program and as an isolation program, are described in detail. Descriptions of one of the more complex isolation programs and of a typical servicing program are given. Pertinent characteristics of the ILLIAC and techniques of fault isolation are also included.**

Introduction

THE MAINTENANCE of an electronic digital computer presents unusual problems for the engineer.[1] A computer is a complex collection of elementary circuits. Although the repair of any individual circuit is simple, the location of the particular circuit at fault among the hundreds of faultless circuits poses a problem of major proportions.

Furthermore, the standard of reliability required is an order of magnitude greater than for other electronic apparatus. Fortunately for the engineer, the computer itself can be used as a versatile test instrument for the localization of faults.

In this paper we discuss first the pertinent characteristics and principles of operation of the Illiac. Next, we describe the typical faults which occur and the effects they have on computer operation. Finally, we discuss the use of three types of diagnostic and servicing programs which enable us to use the computer to diagnose its own troubles. These three kinds of programs answer the questions: Is the computer working correctly? Which part of the computer is at fault? How should this analogue control be adjusted?

Because persistent faults can usually be traced easily with a voltmeter, this paper is concerned mainly with intermittent faults. Refined methods are often required for intermittent faults, especially when the error rate is small.

TABLE I

CHARACTERISTICS OF THE ILLIAC

Computer type	parallel, asynchronous, general purpose
Register capacity	40 binary digits
Memory capacity	1,024 words each of 40 binary digits
Number of tubes	
Memory	900
Arithmetic unit	1,100
Control	600
Input-output	100
Total	2,700
Type of instruction	Single address, two instructions per word
Number of digits defining an instruction	8 binary digits
Number of digits defining a memory position	10 binary digits
Operation times	
Multiplication max.	822 μsec
min.	642 μsec
Division	772 μsec
Addition	72 μsec
Input	4 msec per character
Output (punch)	40 msec per character
Total operation time	3,000 hours[1] (approx.)
Tube failures (excluding cathode-ray tubes)	120[1] (approx.)

[1] On April 20, 1953.

Characteristics of the Illiac

The Illiac, which was completed in September, 1952, is the second automatic electronic computer built at the University of Illinois. It is of the same general type as the Institute for Advanced Study computer at Princeton.[2] In particular, it is a parallel computer with an electrostatic Williams memory. The memory is the only synchronous part of the computer, the rest of the control being asynchronous and designed so that the completion of one operation initiates the next. The computer works internally in the binary system and has 40 binary digits

* Decimal classification: 321.375.2. Original manuscript received by the Institute April 27, 1953.

† University of Illinois, Engineering Research Laboratory, Urbana, Illinois.

[1] Other papers discussing the maintenance of digital computers were published in the 1953 Convention Record of the I.R.E.

[2] Descriptions of digital computers similar to the Illiac are given in: G. E. Estrin, "A description of the electronic computer at the Institute for Advanced Study," *Proc. Assoc. for Computing Machinery*, pp. 95–109, Toronto, Ontario, Canada; September, 1952; and R. E. Meagher, and J. P. Nash, "The ordvac," *Rev. Elec. Digital Computers*, pp. 37–43, 1952. (Proc. of Joint AIEE-IRE Computer Conference; December, 1951).

for a single word or number. The instructions are single-address instructions and packed two in a word. The input unit reads teletype tape by means of a photoelectric tape reader and the output unit punches teletype tape. Some of the Illiac characteristics are given in Table I.

The Illiac Memory

The memory is of the electrostatic Williams type, binary digits being stored as charge distributions on the phosphor of commercially available 3KP1 cathode ray tubes. A digit is read from the memory by sensing the appropriate charged area of the phosphor with the electron beam; the resulting potential change is electrostatically coupled to a wire screen on the outer face of the cathode ray tube and is amplified to the signal level of the logical circuits of the computer.

Such a memory is subject to a variety of faults. First, flaws in the phosphor may make storage marginal or impossible. Second, frequent consultations of one area of the storage surface may affect the digits stored in the immediatevicinity. Third, small noise signals may begenerated in cathode-ray tubes or amplifier circuits, causing errors in stored data. We shall refer to these as flaws, read-around faults, and random faults, respectively.

Susceptibility of the memory to error has affected both the physical structure and the circuit design of the Illiac. Unlike the remainder of the machine, a pluggable chassis is associated with each of the forty digital positions of the memory. Three controls for each of forty cathode-ray tubes are readily available for adjustment. A separate test rack is used for preliminary selection of cathode-ray tubes and for fault isolation within a pluggable chassis.

The Arithmetic and Control Unit

Arithmetic and control units of Illiac are a complex arrangement of a few types of direct coupled logical circuits, circuits in which tubes are used in an on-off fashion. These are best described from a functional viewpoint.

In the arithmetic unit, flip-flop registers are used for number storage; gates are used for number transfers from one register to another. Numbers in two registers are added with a parallel logical adder, subtraction being carried out by using a complement. Halving and doubling is done by shifting numbers right or left. Two registers are used to perform a shift by gating from the first to the second and then back to the first with the digits displaced one position to the right or to the left. Multiplication and division are performed as sequences of additions or subtractions with shifts. Since the computer is a parallel one, corresponding circuits for each digit are activated simultaneously.

In order to localize a fault in the arithmetic unit we have to find both the digital position and circuit involved. Although an arithmetic error may be discovered as a single digit error, it does not always follow that it occurred in the indicated digital position as it may have been shifted before it was discovered.

The control circuits supervise the sequencing required in the arithmetic unit, the selection and execution of instructions, and the use of the memory. A particular circuit is identified by its function, and failures are localized by interpretation of the malfunctioning produced.

The sequencing circuits of the control are designed so that the completion of one operation initiates the next. This allows circuits to operate at their natural speeds and causes certain faults to stop the computer.

The Input-Output Unit

The Illiac is equipped with a photoelectric tape reader, a tape punch, and a teletypewriter. These input and output devices perform mechanical operations under the control of electronic impulses supplied by the computer. Faulty operation results when a mechanical part is out of adjustment.

Detection Programs

The maintenance engineer of an electronic digital computer must be able not only to localize faults quickly when they occur, but he must also be able to minimize the chance of faults occurring during scheduled operation time. For the latter purpose, a stringent program which thoroughly tests all parts of the computer is needed. We call such a program a detection program. The detection program is designed to exercise each component of the computer through all its possible states. Furthermore, the duty cycle has to be high enough so that all parts of the computer are tested under dynamic conditions. Certain circuits of the computer are connected to many other circuits. For example, a clear driver is used to clear simultaneously all the digits of a 40 digit arithmetic register to zeros or to ones. To test this circuit, it is not necessary to try all of the 2^{40} combinations of digits, because it is known that the maximum and minimum load conditions occur as the registers are filled with ones and zeros. Thus, using these special cases, it is possible to test these circuits adequately without trying all the many combinations. It will be noted, however, that such circuits need specially devised tests.

It is almost unnecessary to state that the test program should be designed so that an absolute minimum number of errors escapes detection. For example, in testing multiplication, we must test all of the digits of the double length product and in testing division we must verify that the remainder is correct.

We are at present using Leapfrog III as a detection program on the Illiac and will describe it in some detail. It is called the "leapfrog" because it has been arranged to "leap" through the memory so that the entire memory is tested. The leap is such that each word of the leapfrog occupies every position in the memory for about one second. Thus every memory position is subjected to all of the different intensities of use from the various kinds of storage in the program.

The memory is tested by a "comparison test" which takes place as the leapfrog is moved through the memory. Fig. 1 shows in diagrammatic form how the program is moved through the memory. At any one time, copy 3, which we call the working copy, is using copy 2 as the raw material to manufacture copy 1. As each word of copy 2 is translated to become a word of copy 1 it is also compared with the corresponding word of copy 5. If we trace the history of a particular copy at a given position in the memory, we discover it is manufactured (as copy 1), tested for correctness (as copy 2), used (as copy 3), and again tested (as copy 5). This ensures that the program is always checked before it is used, thus giving the maximum chance that a memory error will be found before it causes the leapfrog to act incorrectly. It also ensures that errors occurring in copies numbered 3, 4, or 5 are also detected.

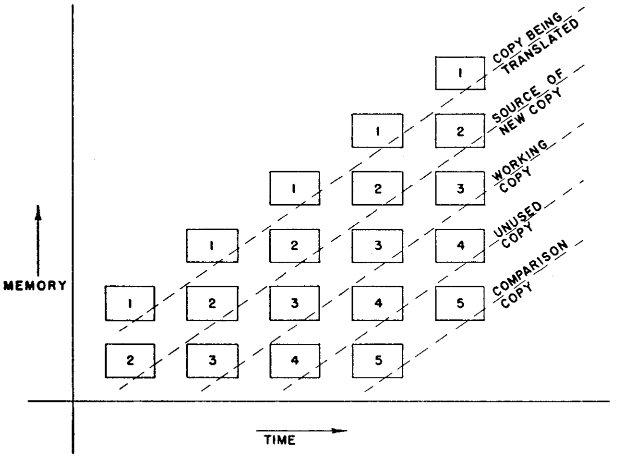


Fig. 1—Motion of the Leapfrog.

The leapfrog contains a stringent arithmetic test. This is split into two parts, a multiplication test and a division test. Both these tests use, and therefore test, other instructions besides multiplication and division. The tests are based upon identities such that all the digits of the numbers involved are checked and any single-digit error will be detected. The numbers used in the arithmetic test are pseudo-random numbers generated from the intermediate results of the previous arithmetic tests. The randomness of these numbers ensures that each digital position of the arithmetic unit is tested under all conditions.

Besides the two tests already mentioned there are additional tests which are performed only once per leap. These additional tests, which are listed in Table II, check certain common circuits of the Illiac under maximum load conditions.

The leapfrog is used to check the serviceability of the Illiac at least twice daily, and is also run during intervals when there is no other demand for computer time. As a result of this policy, and since the leapfrog is more stringent than programs used for calculation, nearly all intermittent faults are first detected by the leapfrog.

TABLE II

INDIVIDUAL TESTS OF THE LEAPFROG*

NAME	EFFECT
Multiplication	A general test of the arithmetic unit, including the use of multiplication instructions.
Division	A general test of the arithmetic unit, including the use of division instructions.
Comparison	Compares copy 2 with copy 5 so that memory errors are detected.
Carry test	Tests the full propagation and collapse of the carry in the adder.
Ones test Zeros test	Tests the functioning of the registers when full of ones or zeros. This essentially tests common driver circuits of the arithmetic unit.
Logical order test	This tests the logical instruction. Every digital position is tested in all conditions.
Shift counter test	This tests every digital position of the shift counter and recognition circuits.
Input-output test	This tests the ability of the input-output unit to read and punch in all digital positions.
Occasional input-output test	This tests the ability of the input unit to ignore certain characters and read correctly a group of characters, and tests the punch while continuously punching.

* Note: The first three tests are done 128 times per leap, the next six tests are done once per leap and the last test once per 128 leaps.

FAULT ISOLATION

When a fault has been detected, it is isolated and repaired as quickly as possible. Unfortunately, there is no simple step-by-step procedure which is applicable for isolation of all types of faults. We shall, however, discuss the isolation procedures applicable to a majority of intermittent failures encountered in operation of the Illiac.

We have noted that the Illiac is composed of three classes of units; the memory circuits, the mechanical parts of the input and output, and the logical circuits of the control and arithmetic unit. The first step in the resolution of a failure is the isolation of the fault to one of these units. Once this is done, a more detailed analysis is needed to define more precisely the location of the fault. The methods of analysis are as varied as the faults.

Our fault isolation methods are based upon the fact that nearly all faults are first detected by the leapfrog. It has been possible to incorporate into the leapfrog many of the features of an isolation program without affecting its stringency as a detection program. In the sections which follow, we describe the isolation features of the leapfrog, and also the more precise localization techniques which are necessary.

The Isolation Properties of the Leapfrog

The first leapfrog test was written for the ORDVAC[1] while it was at the University of Illinois. Although it was

[1] The ORDVAC was built for Army Ordnance by the University of Illinois and has been in operation at the Aberdeen Proving Ground, Maryland, since March, 1952.

a stringent detection test, the ORDVAC leapfrog was unsatisfactory for the isolation of intermittent faults. The reason was that a long time elapsed before some errors were repeated so that it proved desirable to extract as much information as possible from each error as it occurred. The following features of the isolation program have therefore been incorporated in the versions of the leapfrog prepared for the Illiac.

When the comparison test of the leapfrog fails, the corresponding words from copies 2, 3, 4, 5 are printed with their respective memory locations. The intermediate results of the translation are also printed, so that an error of translation can be distinguished from a memory error. If the memory is at fault, the digital position and location of the error can be found from the data.

When the arithmetic test fails, all the intermediate results are printed and the test is automatically repeated. This action of testing and printing continues until the test is satisfied. Thus, if we have an intermittent error, we eventually obtain a correct set of intermediate results. This allows us to determine within one or two instructions the place at which the error occurred. On occasions it has even been possible to determine which step of the multiplication has gone wrong.

Besides these diagnostic features, the arithmetic and the special tests are arranged so that intermediate results used further in the calculation are stored in two memory locations and these are also printed out. This allows us, when one of these tests fails, to say definitely if the memory or arithmetic unit was at fault. When one of the special tests fails, a test identifying number and the intermediate results are printed to enable us to determine the nature of the fault.

The words and numbers are printed out in sexadecimal (base 16) notation, rather than the decimal system, because this is more helpful in diagnosing binary faults. The layout of the printed results has been chosen so that the error is as obvious as possible.

Occasionally a memory fault causes the working copy to become incorrect. In this case a special routine is used to read the leapfrog from the input tape and compare it with the working copy in the memory. The program prints discrepancies so that we can determine the nature and location of the memory fault.

Isolation Procedures

Because the leapfrog is a stringent detection test there are practical limitations to its diagnostic powers. When an error has been detected with the leapfrog further steps are often required to isolate the fault. However, it is relatively simple to find faults in the input, output, or memory circuits from the data supplied by the leapfrog.

Fault Isolation in the Mechanical Parts of the Input-output: Failure of an input-output test of the leapfrog indicates whether the tape reader or punch is at fault. Simple programs which test the faulty mechanism at a higher duty cycle are then used if required. Such failures are generally cured by mechanical adjustment.

Fault Isolation in the Memory: A memory fault is first isolated by the leapfrog to a particular digit of a word in the memory. A cathode-ray oscilloscope is then switched to the chassis of the failing digital position. By inspecting the wave forms displayed, it is usually possible to discover whether the chassis or cathode-ray tube is at fault. If the cathode-ray tube is at fault, the trouble can often be cured by adjusting the controls of the cathode-ray tube; but occasionally replacement of a cathode-ray tube is necessary. Faults which occur in a circuit of a chassis are diagnosed on a separate test rack after the faulty chassis has been replaced by a spare. No attempt is made to isolate a fault within a chassis while it is in the computer.

Fault Isolation in the Control or Arithmetic Unit: Faults in the arithmetic or control unit are usually caused by bad vacuum tubes or faulty connections. Circuits in this part of the Illiac have been conservatively designed, and failures of components other than tubes have not occurred.

Intermittent faults are caused either by shorted tubes, bad solder connections or by marginal circuit operation resulting from tube deterioration. When an intermittent fault is encountered we endeavor to increase the error rate. This can be done by increasing the duty cycle in the suspected part of the computer by a specially designed program written for that purpose. For intermittent shorts the error rate can be increased further by vibrating the suspected part of the computer with a hammer. With the program to give indication of the computer failures, we can discover the element of the computer most sensitive to vibration. If the error is due to marginal operation, then an alteration of a power supply voltage will often cause the error to become persistent, so that it can be traced. If the fault cannot be reduced to a persistent one, then measurements are made in the suspected circuits, either with a voltmeter or with an oscilloscope.

In the arithmetic unit, the technique of interchanging two parallel units is often used to verify other indications during the final localization process.

The effects of faults in the arithmetic unit and in the control are quite different and are usually easy to distinguish. Since each circuit of the control is associated with a control function, a faulty control circuit can be traced by careful observation of the effects of the malfunctioning. In the arithmetic unit, a faulty circuit initially causes a single digit error; the error may, however, be propagated before it is detected in such a way that the circuit at fault is difficult to find.

The computer is used as a versatile test instrument for localization of some intermittent control faults and for nearly all intermittent arithmetic unit faults. Usually the fault is detected by the leapfrog so that some localization clues are available; for example, it may be known that the fault caused an error during a multiplication. For the more difficult faults, a sequence of isolation programs is used, each successive program being shorter and applicable to a smaller part of the computer than the previous one. The progtams used in the final localization of the faults are usually very short and are written on the spot, being discarded when the fault is found. Often the program can be reduced to two instructions, using a special mode of operation of the Illiac known as "instruction pairs." In this mode of operation two instructions are set up on a forty digit flip-flop register and obeyed alternately, with continually increasing addresses (modulo 1024).

The Programmed Multiplication Test: The isolation programs used for localization of faults are as varied as the faults themselves. An example of one of the more complicated isolation programs is the programmed multiplication test, designed to localize faults causing errors during a multiplication.

The Illiac performs a multiplication as a sequence of right shifts and additions. During the multiplication, the duty cycle is high in many parts of the arithmetic unit, and some highly intermittent faults occur only during multiplication.

A multiplication can be characterized as a two dimensional array of digits as shown in Fig. 2a. In a parallel machine one of these dimensions is the digital position in the registers and the other is the step of the multiplication (or time).

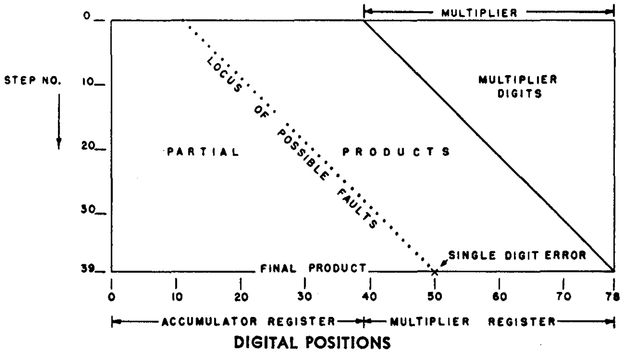


Fig. 2a—Multiplication with single digit error in product.

From a single digit error in a product we can find the relationship between the digital position at which the error may have occurred and the corresponding step. This relationship is represented by the dotted line of Fig. 2a. Unfortunately, not much information is given about the position or step at which the error occurred. The programmed multiplication test is used to isolate such a fault.

The test consists of two parts. The first part is a programmed multiplication and is used to generate in tabular form the two dimensional array of digits. The second part is a series of 39 partial multiplication tests, each of these tests splitting a single multiplication into two partial ones; the first simulates the initial n steps and the second simulates the last $(39-n)$ steps of the multiplication. The tabular values of the programmed multiplication are used for comparison with the final values of the first partial multiplication and are used as initial conditions for the second one.

When a partial multiplication test fails, information is provided for fault localization as indicated in Fig. 2b. An error resulting from a single faulty digital position may be detected in any of 39 digital positions of the product. On the other hand, if the error is detected by a partial multiplication test the range is reduced as shown on the diagram.

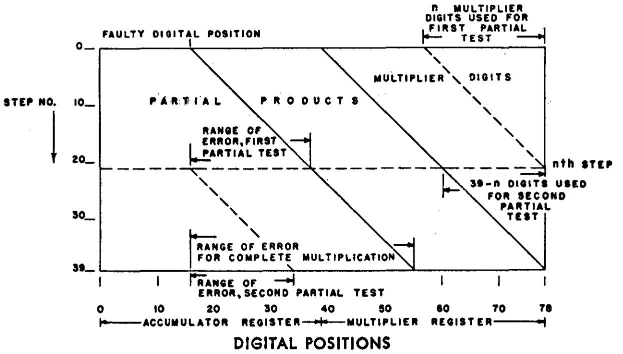

FAULTY DIGITAL POSITION
n MULTIPLIER DIGITS USED FOR FIRST PARTIAL TEST
STEP NO.
0
10
20
30
39
MULTIPLIER DIGITS
PARTIAL
PRODUCTS
RANGE OF ERROR, FIRST PARTIAL TEST
nth STEP
39-n DIGITS USED FOR SECOND PARTIAL TEST
RANGE OF ERROR FOR COMPLETE MULTIPLICATION
RANGE OF ERROR, SECOND PARTIAL TEST
0
10
20
30
40
50
60
70
78
ACCUMULATOR REGISTER
MULTIPLIER REGISTER
DIGITAL POSITIONS

Fig. 2b—Fault localization with partial multiplication tests.

Sometimes an oscilloscope is needed for the final localization of the fault. The oscilloscope presents a cross section of the two dimensional array of digits, that is, a sequence of digits at a particular digital position. This display can be compared with the appropriate column of the table calculated by the programmed multiplication test, thus helping in the final stages of the isolation of the fault.

Computer Maintenance

Occasionally certain controls have to be altered to keep the computer in good adjustment. Such adjustments pertain to the memory, input, or output unit, as the rest of the computer is designed using components in on-off circuits. A typical adjustment is the optimization of the read-around ratio using the intensity control of a cathode ray tube.

To facilitate such adjustments, servicing programs have been written. The programs supply appropriate test conditions to the unit being adjusted and detect malfunctioning. A typical servicing program is the read-around adjustment program. This program continually scans the memory for points of low read-around ratio and indicates these points. If an adjustment is made while this program is running, the effect on the read-around ratio is easily seen, so that the optimum control setting can be discovered by trial and error.

In order to prevent gradually deteriorating components from causing errors during the regular scheduled operating time, marginal tests are performed periodically. There is no marginal testing equipment built into the Illiac, and the tests are performed by altering the power supply voltages, ac and dc. The leapfrog is used to detect the point at which a circuit fails, the printed error indication being kept as a record. When the tolerance range of any voltage becomes too small for satisfactory performance, the components causing the trouble are localized and replaced.

During the past nine months much more trouble has been caused by shorted tubes and open filaments than slowly deteriorating tubes so that few failures have been prevented in this manner. However, as the Illiac ages, these marginal tests should prove more valuable.

Conclusion

Diagnostic and servicing programs are essential for the efficient maintenance of an automatic digital computer. Since the engineer's knowledge of the computer and the power of the test programs are functions of one another, one would expect the diagnostic and servicing programs to be continually improved, especially with a new computer. It is our opinion that only by frequent and intensive searches for the weak elements, can one maintain the degree of reliability required for a truly serviceable computer, and furthermore, the degree of reliability which can be maintained is determined by the power of the test programs.

Acknowledgment

The material presented in this paper describes the joint efforts of the staff members of the Digital Computer Laboratory of the University of Illinois. The authors are especially indebted to Professor J. P. Nash of the Computer Laboratory for his many helpful suggestions during preparation of the paper.

Odd Binary Asynchronous Counters*

J. E. ROBERTSON†

Summary—**This paper describes a general method for modifying conventional binary asynchronous counters such that the counting register advances by any desired odd integer for each received count. The pertinent design features of conventional additive and subtractive asynchronous counters are reviewed. Simplification of the design of a counter which advances by an odd integer is achieved through use of a set of alternately additive and subtractive subcounters. An example of the logical design of a counter which advances by 13 is presented.**

Introduction

An unusual type of counter described by Ware[1] and Brown[2] is used in binary asynchronous computers patterned after the Institute for Advanced Study computer. Such counters employ a pair of gating operations which are mutually exclusive timewise to advance the state of a set of "true" toggles by one unit. This paper describes a general method for modifying such counters so that the state of the set of "true" toggles is advanced by any desired odd integer by each pair of gating operations.

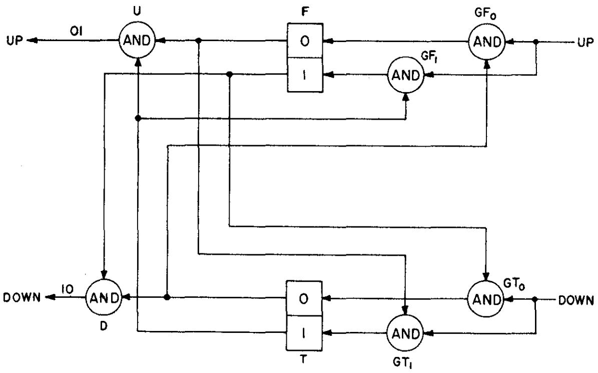


Fig. 1—Logical equivalent of one stage of an adding asynchronous binary counter.

Review of Counter Design Principles

One stage of the asynchronous counter[3] consists of a "true" toggle, a "false" toggle, circuits for gating the toggles to their allowed states, and circuits for forming a pair of gating levels to operate the next most significant stage. The logical equivalent of one stage of the asynchronous counter is shown in Fig. 1. The effect of the UP input gating operation is to set the false toggle F to agree with the true toggle T. For example, if $T=0$ *and* an UP input gating operation occurs, F is set to 0 by the "and" circuit GF_0. Similarly, the effect of the DOWN gating operation is to set the true toggle T to disagree with the false toggle F. The true and false toggles thus proceed through a Gray code sequence of states as alternate UP and DOWN input gating operations are applied (Table I). The UP and DOWN gating levels for the next most significant stage are formed by sensing the states 0 1 and 1 0, respectively, with the "and" circuits U and D. For a counter whose true toggles are to assume the sequence of states corresponding to the binary integers 0 to 2^{n-1}, n stages of the type illustrated by Fig. 1 are required. The process of counting in a two stage counter is indicated in Table II. It is

TABLE I

Sequence of States of True and False Toggles of One Stage

Input pulse	F	T
	1	0
up		
	0	0
down		
	0	1
up		
	1	1
down		
	1	0

TABLE II

Sequence of States of Toggles in a Two Stage Adding Asynchronous Counter

State of true toggles		Most significant stage		Least significant stage		Input gating operation
		F	T	F	T	
0	0	1	0	1	0	
						up
				0	0	
						down
0	1	0	0	0	1	
						up
				1	1	
						down
1	0	0	1	1	0	
						up
				0	0	
						down
1	1	1	1	0	1	
						up
				1	1	
						down
0	0	1	0	1	0	
						up
				0	0	

* Original manuscript received by the PGEC, August 18, 1955; revised manuscript received, November 10, 1955.

† Digital Computer Lab., Univ. of Illinois, Urbana, Ill.

[1] Willis Ware, "The logical principles of a new kind of binary counter," Proc. IRE, vol. 41, pp. 1429–1437; October, 1953.

[2] R. M. Brown, "Some notes on logical binary counters," Trans. IRE, vol. EC-4, no. 2, pp. 67–69; June, 1955.

[3] The asynchronous counters discussed here have been called "self-instructed counters" by Ware.

to be noted that the sequence of states assumed by the true toggles is that of a conventional binary counter.

As has been pointed out by Ware and Brown, the adding asynchronous counter can be converted to a subtracting counter by interchanging the states used to form the UP and DOWN gating levels for the next most significant stage (Fig. 2). For this arrangement, the sequence of states is that shown in Table III, and the true toggles assume states corresponding to a descending sequence of binary integers.

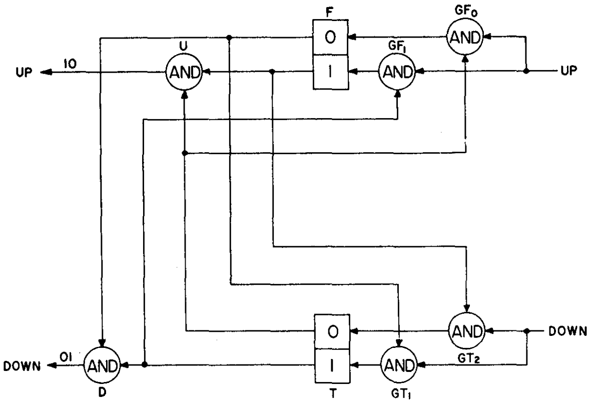


Fig. 2—Logical equivalent of one stage of a subtracting asynchronous binary counter.

TABLE III
SEQUENCE OF STATES OF TOGGLES IN A TWO-STAGE SUBTRACTING ASYNCHRONOUS COUNTER

State of true toggles		Most significant stage		Least significant stage		Input gating operation
		F	*T*	*F*	*T*	
0	0	0	0	1	0	up
				0	0	down
1	1	0	1	0	1	up
				1	1	down
1	0	1	1	1	0	up
				0	0	down
0	1	1	0	0	1	up
				1	1	down
0	0	0	0	1	0	up
				0	0	

DESIGN OF AN ASYNCHRONOUS COUNTER WHICH ADVANCES BY AN ODD INTEGER

In some applications, it is desirable that an n-stage counter assume a sequence of states such that successive true states differ by an odd integer other than one. The counter designs described in the previous section can be easily modified for any specified odd integer q. The modification requires that the counter be subdivided into subcounters, with the subcounters alternately additive and subtractive.

For an n-stage counter, the stages are numbered 0, 1, 2, $\cdots$, $n-1$, where stage 0 is the least significant one. We then express q as an alternating series of powers of two. If q is in the range $-2^{n-1}<q<2^{n-1}$, then

$$q = \pm 2^{p_{k-1}} \cdots \mp 2^{p_3} \pm 2^{p_2} \mp 2^{p_1} \pm 2^{p_0}$$

with $0=p_0<p_1<p_2<p_3< \cdots <p_{k-1}<p_k=n$. The p_i are integers uniquely determined by the value of q. For example, if $q=13$, $q=+2^4-2^2+1=16-4+1$. The number of terms (k) in the series representation of q determines the number of subcounters. The sign of the term determines whether the corresponding subcounter is an adding or subtracting subcounter. The exponents p_i determine the number of stages in each subcounter. Specifically, sub-counter i ($i=0, 1, \cdots, k-1$) has $p_{i+1}-p_i$ stages numbered $p_i, p_i+1, \cdots, p_{i+1}-1$. If the sign of the term 2^{p_i} is positive, subcounter i is an adding subcounter; if the sign is negative, subcounter i is a subtracting one. For example, if $q=13$ and $n=5$, then $k=3$, and $p_0=0$, $p_1=2$, $p_2=4$. Three subcounters are:

Subcounter 0	stages 0 and 1	additive
Subcounter 1	stages 2 and 3	subtractive
Subcounter 2	stage 4	additive

The alternation of adding and subtracting subcounters simplifies the counting process when a carry (or borrow) is transmitted from one subcounter to the next most significant one. The input DOWN gating operation is applied to the least significant stage of each subcounter, except that it is inhibited by a carry (or borrow) from subcounter $i-1$. One requirement for the carry of an adding subcounter is that its true toggles are in the 1 state; similarly, a requirement for the borrow of a subtracting subcounter is that its true toggles are in the 0 state. The rules for inhibition of the UP gating operation require some elaboration. Suppose that subcounter i is an additive one and that all toggles of subcounter i are in the one state. Then the addition of a one to stage p_i should produce a carry whose effect is the addition of a one to stage p_{i+1}, the normal subtraction and the additive carry from stage p_i cancel one another; the net result is that we inhibit the UP gating operation, thereby preventing any change of the true toggle of stage p_{i+1}. In short, the conditions for a carry (or borrow) from a given subcounter are that all toggles are in the carry (or borrow) state and that an UP input is applied to its least significant stage. The effect of a carry (or borrow) from a given subcounter is to prevent an UP input from being applied to the next most significant subcounter. A block diagram of the counter is shown in Fig. 3.

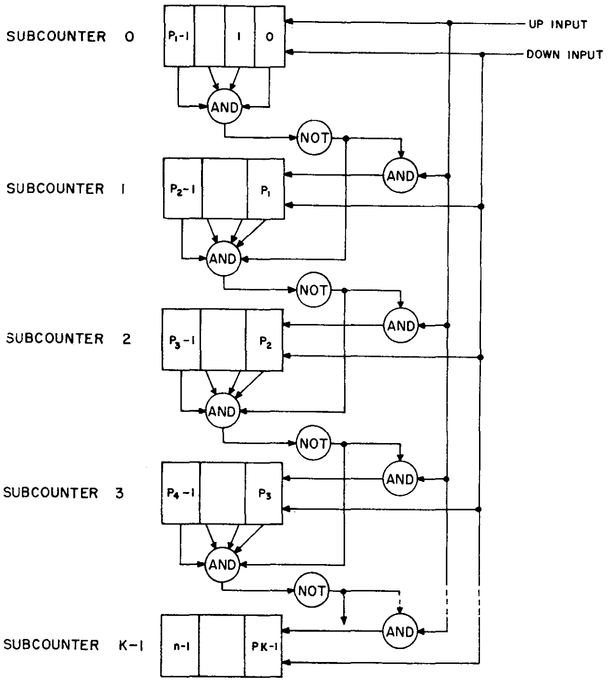


Fig. 3—Block diagram for an odd binary asynchronous counter.

The simplification achieved by use of the alternating series representation of q may be more apparent if we consider the design difficulties when the conventional positive series representation is employed (*e.g.* $q=13$, $=2^3+2^2+2^0$). Since all subcounters are additive, it is necessary when a carry arises in subcounter $i-1$ to initiate two counting operations at the least significant stage of subcounter i. Not only would the circuitry for generating an extra pair of gating operations be complicated, but additional time for the operations would be required as well.

Illustrative Example

The following example arose in connection with the design of a drum storage unit for the Illiac. It was desired that a set of 32 words comprising one track should be interlaced in such a way that for every fifth word, the count contained in the true toggles increased by 1. Inspection of Table IV reveals that an equivalent requirement is that the state of the true toggles should increase by 13 for each word. The equivalence may also be deduced from the fact that

$$5 \times 13 \equiv 1 \text{ modulo } 32.$$

As noted in the previous section $q=13$, $n=5$, $k=3$, $p_0=0$, $p_1=2$, $p_2=4$.

Three subcounters are required, as follows:

TABLE IV

Sequence of True States for Counter which Counts by 13's

0	0	0	0	0
0	1	1	0	1
1	1	0	1	0
0	0	1	1	1
1	0	1	0	0
0	0	0	0	1
0	1	1	1	0
1	1	0	1	1
0	1	0	0	0
1	0	1	0	1
0	0	0	1	0
0	1	1	1	1
1	1	1	0	0
0	1	0	0	1
1	0	1	1	0
0	0	0	1	1
1	0	0	0	0
1	1	1	0	1
0	1	0	1	0
1	0	1	1	1
0	0	1	0	0
1	0	0	0	1
1	1	1	1	0
0	1	0	1	1
1	1	0	0	0
0	0	1	0	1
1	0	0	1	0
1	1	1	1	1
0	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	0	1	1

Subcounter 0, additive, stages 0 and 1

Subcounter 1, subtractive, stages 2 and 3

Subcounter 2, additive, stage 4

The logical diagram of the desired counter which counts by 13's is shown in Fig. 4 (opposite). For the physical modification of an existing adding binary counter, the U and D "and" circuits (Figs. 1, 2) of stages 1 and 3 were used for forming the UP inputs to stages p_1 and p_2 (*i.e.:* 2 and 4), so that only $2\frac{1}{2}$ additional tubes were required for the modification. The approach initially considered involved use of a translation circuit requiring 15 tubes.

Conclusions

This paper presents a solution to a problem which was, to the author, somewhat difficult conceptually but simple in implementation. Although the description here applies specifically to asynchronous counters counting by odd integers, several extensions of the approach appear possible. An obvious extension is to counting by even integers. If an n-stage counter advancing by 2^pq, with q an odd integer, is desired, it is sufficient to design a counter which advances by q in the $n-p$ most significant digits, with the p least significant digits always zero. It is thus possible to design a counter which advances by any integer, even or odd. It seems reasonable that the approach can also be employed in the design of other types of binary counters.

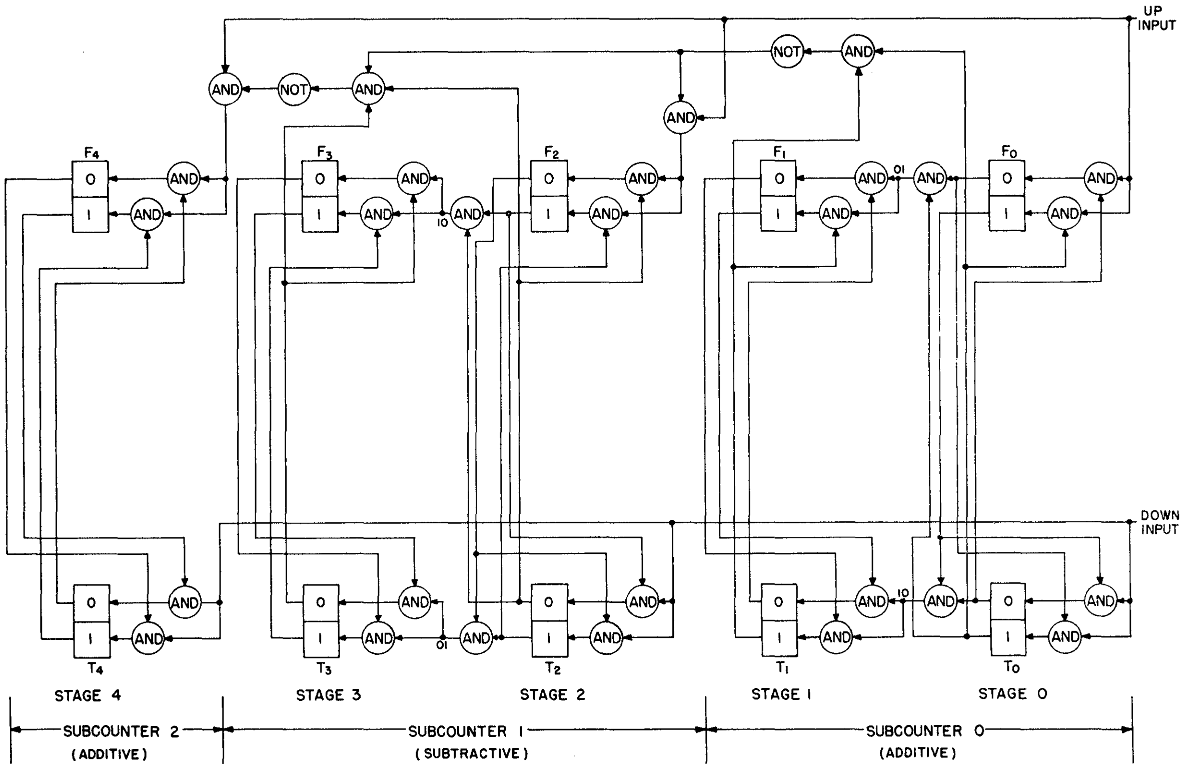


Fig. 4.—Logical diagram of counter which counts by 13's.

An early paper on the ORACLE

Introduction *by* MARGARET K. BUTLER
Argonne National Laboratory
Argonne, Illinois

The engineering design of the ORACLE, the second general-purpose computer built at Argonne National Laboratory, is described in the paper reprinted here. It was presented originally at the meeting of the Association for Computing Machinery held at the University of Toronto, Ontario, Canada, September 8–9, 1952.

This paper is of interest as illustrative of the computer development carried out at Argonne National Laboratory. The author, J. C. Chu, supervised the engineering and construction work there that produced both the AVIDAC and the ORACLE. In this paper he sets down engineering design principles followed, describes the general characteristics and the logical design of the second machine, and provides several illustrations of engineering circuits used. Of primary interest, however, is the discussion of the features of the ORACLE memory, which represented a significant advance in Williams-tube systems.

The ORACLE memory was composed of 80 cathode ray tubes, packaged two per unit, which were designed to check one another in 1024-word mode or to operate independently as a 2048-word memory. In addition, a circuit was incorporated to increase the reliability of the Williams-tube memory by automatically stabilizing the beam currents in the cathode ray tubes. One storage location in each tube was used as a test point. The circuit was designed to sample the output of this location periodically and adjust the beam current to maintain this output at a constant amplitude. Since all the other storage locations of the tube used the same beam, their outputs were also stabilized.

The first name given the Oak Ridge machine, ORAC, is reflected in the title of the paper; in September 1952, the name was changed to ORACLE.

THE OAK RIDGE AUTOMATIC COMPUTER

By

J. C. Chu
Argonne National Laboratory

THE ORACLE

The ORACLE (Oak Ridge Automatic Computer Logical Engine) is a general purpose digital computer, designed and constructed at the Argonne National Laboratory for the Oak Ridge National Laboratory. Production of the major electronic units has been carried out by Technitrol Engineering Company of Philadelphia, Pennsylvania. The construction started in November, 1951. In logical design, it belongs to the parallel asynchronous type. The number system is binary with fixed binary point. The orders which can be executed by the machine can be classified as of the following types:

Arithmetic
Memory and arithmetic unit transfer
Control transfer
Memory and auxiliary memory transfer.

A total of seventy-seven individual orders is provided.

In engineering design, there are two principles worth mentioning. As to circuitry and components, we believe that each type of circuit and circuit component has merits as well as shortcomings. Each such element should be used according to individual circumstances. For this reason one will find in the ORACLE DC as well as AC coupling circuits, synchronized as well as asynchronized circuits, logical as well as analog gating circuits, crystals as well as vacuum tubes, triodes as well as pentodes, and receiving as well as transmitting tubes.

Both speed and reliability are essential. Features that make for reliability are rigorous logical design that minimizes the number of components required, allowance in every circuit design of ample tolerances on components, and strict adherence to the principle that, under no circumstances, is speed to be obtained at the expense of lowered standards of reliability. No internal checking of any kind is employed throughout.

The three-dimensional chassis layout of the JOHNIAC is used in the arithmetic and control unit; plug-in subassemblies are used in the memory unit.

Figure 1 summarizes the general characteristics of the ORACLE. It shows that the number of digits* assigned to order is nine and the number assigned to address is eleven. A complete command, therefore, is composed of twenty digits. A word is composed of forty digits,† so a word is made of two commands.

Also, in Figure 1, it is shown that the memory system of the ORACLE can be operated in either of two modes, namely Mode 1024 or Mode 2048. This will be explained further in the latter part of this paper. It should also be noted that a total of five main voltages is used.

Modified teletype equipment is used as the input and output unit with a read-in speed of 24 characters per second and punch-out speed of 15 characters per second. Multiple channel magnetic tape with a drive using accumulation of the tape in baskets is planned as the auxiliary memory unit.

The remaining part of this paper contains a brief review of the logical design of the ORACLE and a few illustrative engineering circuits.

Figure 2 is a diagram of the ORACLE's logical symbols. In the logical diagrams, no attempt is made to achieve precise conformity with the engineering diagrams. Signal polarity, type of gating circuit and drivers are not presented in the logical diagram. The signal on each line is either a "1" or a "0." It is a "1" when the line is activated. The active state is marked along the line.

Figure 3 is the arithmetic and control logical diagram. The dotted lines are the information lines and the solid lines are the control lines. The A and Q registers have five tubes per stage. These are marked as "A" and "Q" on the diagram. Asymmetrical toggles are used. These have a maximum shifting speed of approximately 4 sec. per shift. There are five tubes per stage in the S-register. Symmetrical toggles are used here.

The plusser or adder, which is indicated as "P" on the diagram, has eighty tubes per stage. It is of the logical adder type with special arrangements for speeding up the carry propagation. The time for carry propagation and collapse is about 4 μseconds.

The static programmer is, in effect, a partially decoded matrix table. It has about one hundred tubes. The set up speed is approximately 5 μseconds.

The dynamic programmer which generates the pulse routine to program the shift registers contains sixty tubes.

The shift counter is a 6-stage binary scaler of the cathode trigger

*It would be more appropriate to use the word "bit" instead of the word "digit".

†Sée the paper of C. L. Perry on the logical design of the computer in this publication.

type. RC differentiating coupling networks between stages are used. The total carry propagation lasts about 0.1 to 0.2 μseconds.

The purpose of the dispatch counter is to dispatch the address specified by the order to the memory, keep account of the progress of the execution of the orders and keep track of memory regeneration. The dispatch counter is composed of the dispatch counter, control counter and regenerator counter which are marked as DC, CC and RC respectively, and consists of nine tubes per stage with a total of eleven stages. The eleventh stage is used only when the memory is operating in the 2048 mode. The half adders used here between stages are of logical and amplitude combination type. In a counter of this type, the set-up time is essentially the gating time. Because the counting rate is relatively slow, the idle time of the counter is used for carry propagation.

The MOSC—memory order sequence control—is the interplay unit between the memory control and the arithmetic control. It serves to convey information between the two units. It has about eight tubes.

As stated before, no typical circuits are set as standards in the ORACLE. The engineering design of some individual units may be of interest. To limit the length of this paper, only the shift register and its drivers and the general principle of the ORACLE memory system will be discussed.

The shift register contains an upper and lower bank, each with forty toggles. The upper and lower toggles illustrated in Figure 4 are typical of the toggles in the register. The four vacuum tubes located between the toggles are the drivers and the 30K, 40K resistors constitute the gating circuits to transfer the information between the upper and lower banks. T^u, T_r, T_d, and T_l are, respectively, the transfer-up gates, transfer-right gate, transfer-down gate and transfer-left gate. There are two voltage levels that accompany each gating signal. The voltage level at which the gate is activated is indicated by an arrowhead. The transfer-in gating circuit is similar to the transfer-up circuit; it is not shown on this diagram.

Clearing the toggles is done by lowering the plate voltage of the left-hand tube, thus lowering the grid of the right-hand tube to cause it to cut off, and the left-hand tube to become conducting. The transfer is done by raising the common cathode of the toggle to cause the left tube to cut off. When the cathode voltage is again lowered, the voltage of the right grid is higher than that of the left grid. The right tube now is then conducting and the left one is cut off. The wave form of the clear and transfer pulses is shown in the same diagram. The minimum clearing time is about 1 μsec. and the transfer time is 1.2 μsec. Clear and transfer start at the same time but transfer should outlast clear by at least 0.2 μsec.

The toggle has a safe margin of stability for variations of the components of twice that specified by the manufacturer and changes of tube characteristics of 50% resulting from aging.

Figure 5 is a plot of plate current (I_p) vs. plate to cathode voltage (I_{pk}) with constant cathode current (I_k) as a parameter. The load line is plotted according to the equation $I_p R_L + e_{pk} = E_{bb}$ and "0" is the operating point of the asymmetrical toggle. From the load line, it is seen that when the left-hand side tube of the toggle is conducting, it draws a plate current of about 4.8 milliamperes with a grid current of 2.7 milliamperes. The plate swing of this tube is from about 12 volts to about 85 volts. Since the plate swing is at a relatively low DC level, it is possible to use a low voltage bleeder ratio so that the grid of the right-hand side tube sees 88% of this plate swing.

Figure 6 shows the clear and transfer driver circuits for the asymmetrical toggles. In order to minimize fluctuation of the drivers' voltage, self-regulating drivers are used. For both the clear and transfer drivers, only those voltage levels which are important for the reliability of the toggle are regulated. These drivers are designed to be consistent with the requirements for feedback stability. They have a rise and fall time approximately 0.2 μsec. with 1% overshoot. The regulation for full load to no load is about 1%.

In the transfer driver circuit, tube 5687 is the driver switching tube. When it is conducting, the regulating amplifier is cut off and the driver output is at −100 volts. If the switch tube 5687 is cut off, then the regulating tube 12AT7 is allowed to function. The top (−13.5) of the transfer pulse is, accordingly, regulated. One circuit of interest is the unit gain electronic voltage divider. It is composed of two 12AT7's in cascade. The lower tube, with a 24K resistor in its cathode, is a constant current tube because the plate current through this tube is entirely dictated by its cathode voltage and cathode resistor. Consequently, whatever voltage swing within the linear characteristics of the tubes occurs at the cathode of the top tube, it will be seen at the plate of the lower tube. The two 22K resistors in series give the required voltage levels shift.

Figure 7 is the ORACLE memory. There are two operating modes available in the ORACLE memory system. In either case, dot-dash display is used. The dot is used to regenerate a zero, and dot-dash is used to regenerate a one. Mode 1 is the 1024-word mode in which time multiplex is used between a pair of Williams tubes to determine the stored information for each bit. In this arrangement, when either tube reads a "1" signal, a "1" is replenished to both. This method overcomes the most common type of screen blemish which would prevent storage of a "1" (dot-dash). Mode 2 is the 2048-word mode in which each tube stores 1024 bits. The first tube is regenerated in the first half of a major cycle and the second tube in the second half of the major cycle. As can be seen from the block diagram, no extra equipment is added to have Mode 1 available in the ORACLE memory system. It has been verified experimentally that when the ORACLE memory is in the 1024 mode, there is no noticeable effect on the repetitive consulting number (or read-around ratio). The 3JP1 tube is selected for the memory; it is expected to give a repetitive consulting number around 500. This number is obtainable because it is possible to use a higher accelerating potential to obtain a better focus; there being no trouble resulting from the common screen blemishes.

Another feature of the ORACLE memory is the ABS* used to stabilize the beam current of the memory tube. Since the "1" signal (signal due to dot-dash configuration) depends on the beam current, a single address (zero location) is reserved for the storage of the "1" signal. The ABS clock can be set from approximately a single memory major cycle frequency to many cycles. It requires a maximum of three memory minor cycles to effect an ABS cycle. One minor cycle is required for synchronization, to complete the memory cycle which has already begun, and to prevent further order execution. When the MOSC and the Williams tube pulse routine generator are locked in this manner, the dispatch counter is automatically cleared to zero and no transfer from RG or DC is possible. Zero address is, therefore, most conveniently chosen for ABS storage. Since the "1" signal stored in the zero address location at the previous ABS cycle may be influenced by the interim storage display, a second memory cycle is used to write a fresh "1" signal at the zero address location and the third minor cycle is used to read the "1" signal. The feedback loop is completed through the amplifier to the ABS integrating circuit which controls the cathode bias of the memory tube.

The effectiveness of the ABS system has been verified by experimentally changing the amplifier gain and the memory tube heater voltages while the memory is artificially programmed for continuous repetitive consulting. For ± 10% variation of heater voltage and reasonable variation of amplifier gain, there was no effect on the RC number when the ABS was on.

Figures 8 and 9 show the physical construction of the basic memory plug-in unit. On top of the base unit are the amplifier and the discriminator. A total of twenty-three vacuum tubes is used to store 2048 bits of information.

* Automatic Beam-Current Stabilization.

A further discussion of the ORACLE memory system is presented in another paper entitled "The Selection of Tubes for the Williams Memory" by J. C. Chu and R. J. Klein in this publication.

The present status of the ORACLE is as follows: Its arithmetic and control unit have been constructed and passed the specified acceptance tests. A model of the memory has been constructed and tested thoroughly for effectiveness. The ORACLE is due for operation the early part of 1953.

ACKNOWLEDGMENT

The author wishes to acknowledge the encouragement given by Dr. Louis A. Turner, Director of Physics Division, Argonne National Laboratory, in carrying out this project and for his assistance in the preparation of this manuscript.

J. C. Chu

OAK RIDGE COMPUTER CHARACTERISTICS:

MACHINE TYPE	PARALLEL	
REGISTER CAPACITY	40	
ALLOWED ADDER CARRY TIME	5	μ.s.
ALLOWED TIME PER SHIFT	6	μ.s.
MULTIPLICATION TIME (eir = 0)	240	μ.s.
MULTIPLICATION TIME (eir = $1-2^{-39}$)	440	μ.s.
DIVISION TIME	440	μ.s.
NUMBER OF DIGITS ASSIGNED TO NUMBER	40	
NUMBER OF DIGITS ASSIGNED TO ORDER	9	
NUMBER OF DIGITS ASSIGNED TO ADDRESS	11	
MEMORY MINOR CYCLE OR MAXIMUM ACCESS TIME	20	μ.s.
MEMORY MAJOR CYCLE		
MODE 1024	20.48	m.s.
MODE 2048	40.96	m.s.
NUMBER OF VACUUM TUBES	3000	

Figure 1—Characteristics of the Oak Ridge Computer

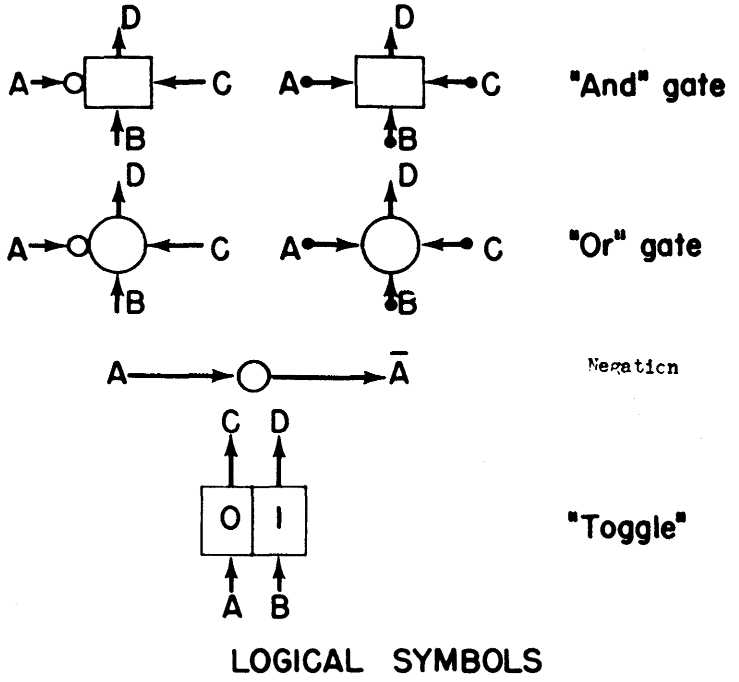
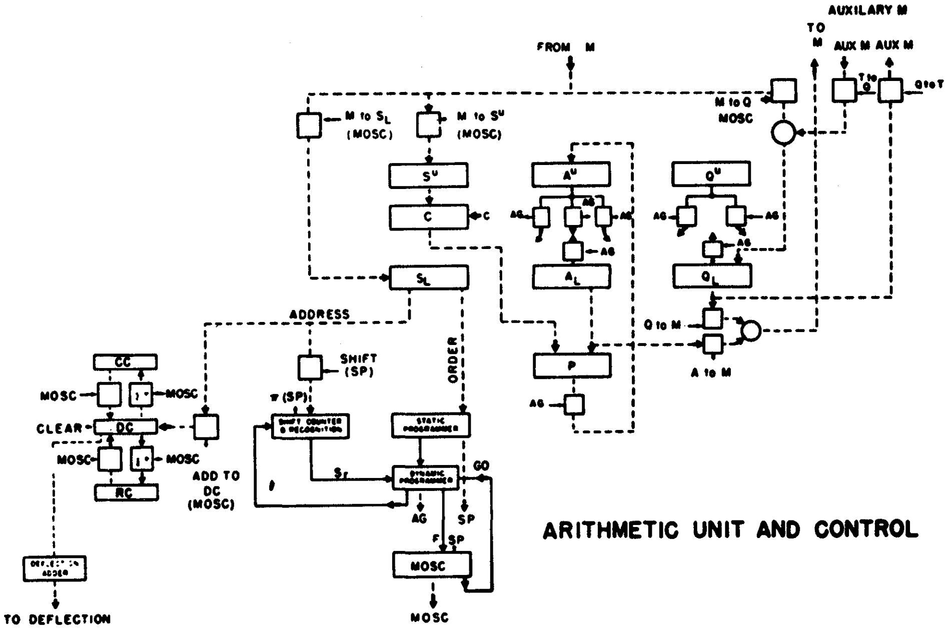


Figure 2—Logical symbols



NOMENCLATURE

S^u	Storage register, upper bank	SP	Static programmer
S^l	Storage register, lower bank	AG	Arithmetic gate
A^u	Arithmetic register, upper bank	SR	Shift recognition
A^l	Arithmetic register, lower bank	π	Multiplication or division order
Q^u	Quotient register, upper bank	M	Memory
Q^l	Quotient register, lower bank	T	Magnetic tape or teletype tape
P	Plusser or adder	ADD	Address
Mosc	Memory order sequence control		

Figure 3—Arithmetic unit and control

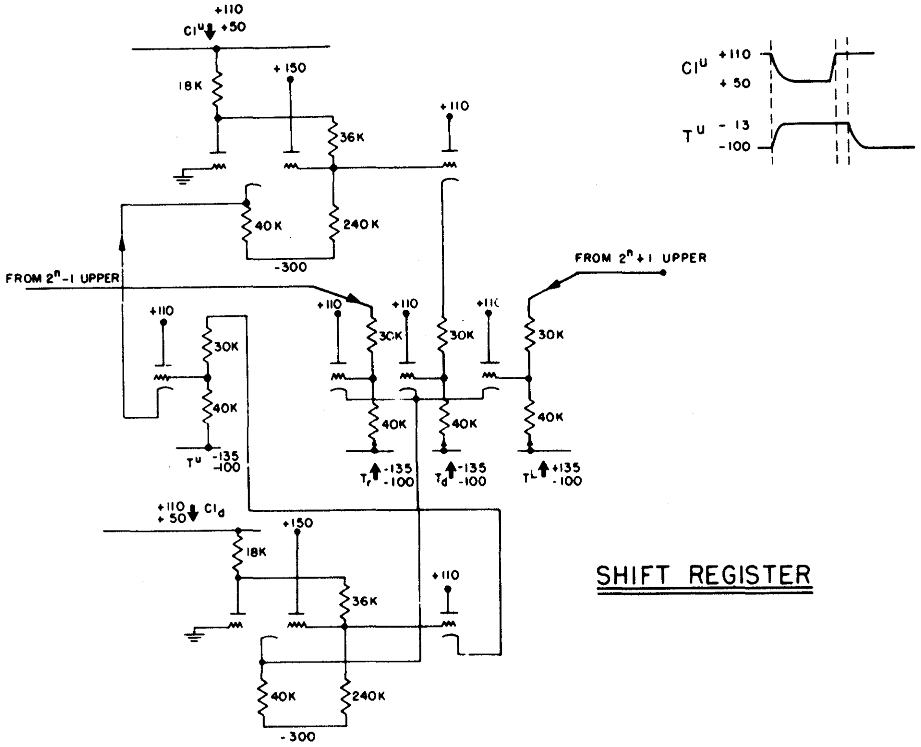


Figure 4—Shift register

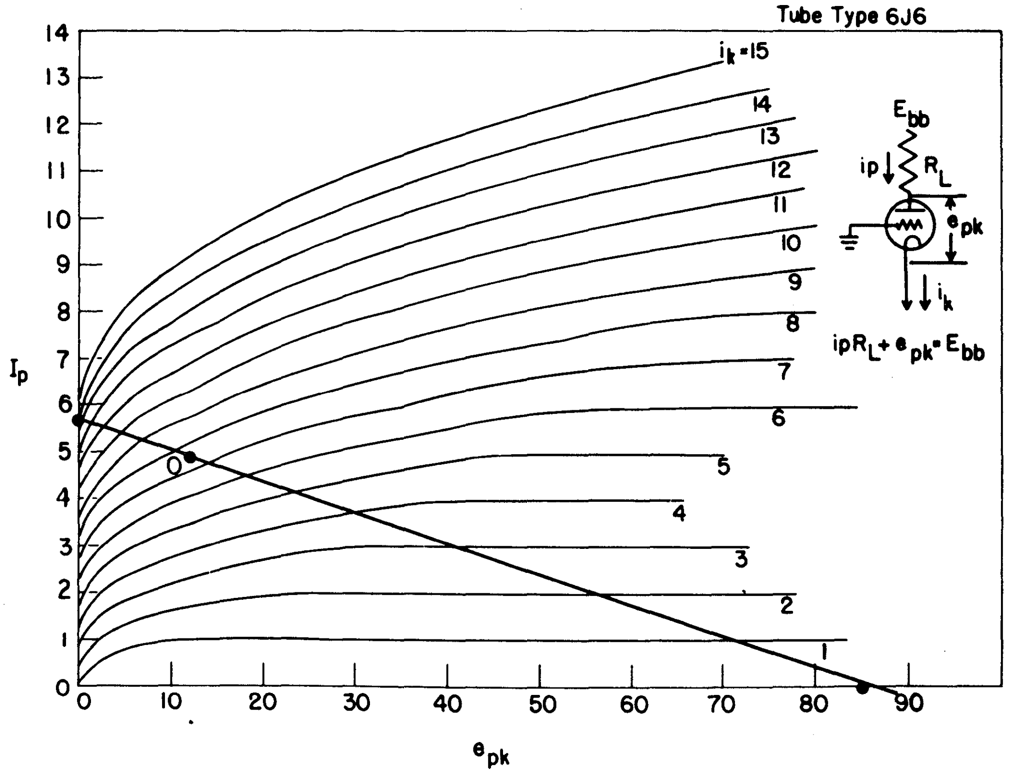
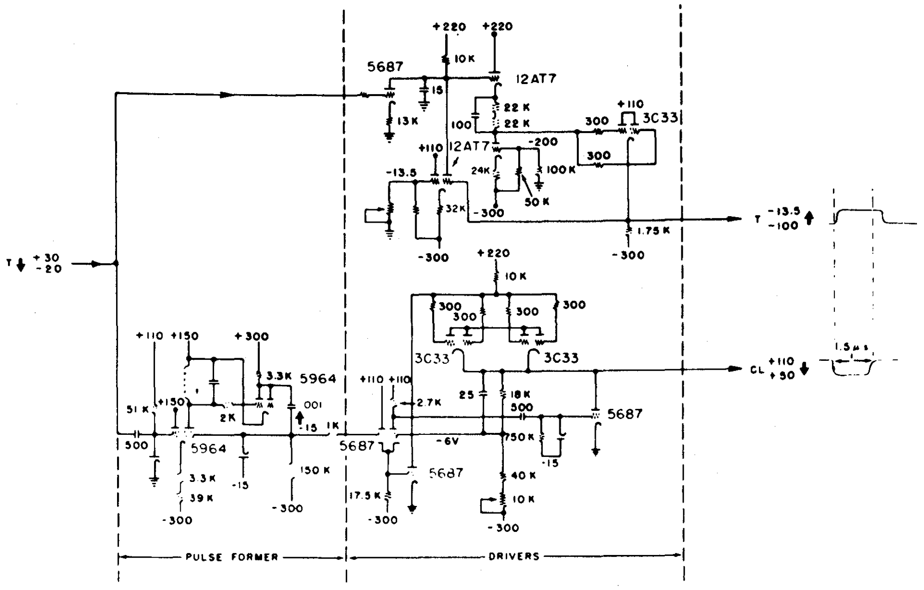


Figure 5—Load lines for the asymmetrical toggle



CLEAR AND TRANSFER DRIVER

Figure 6—Clear and transfer driver

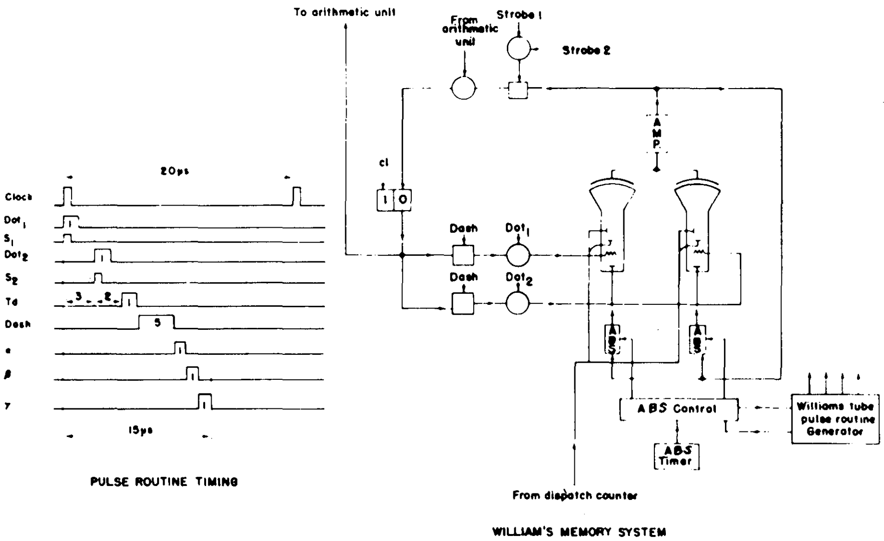


Figure 7—ORACLE memory system

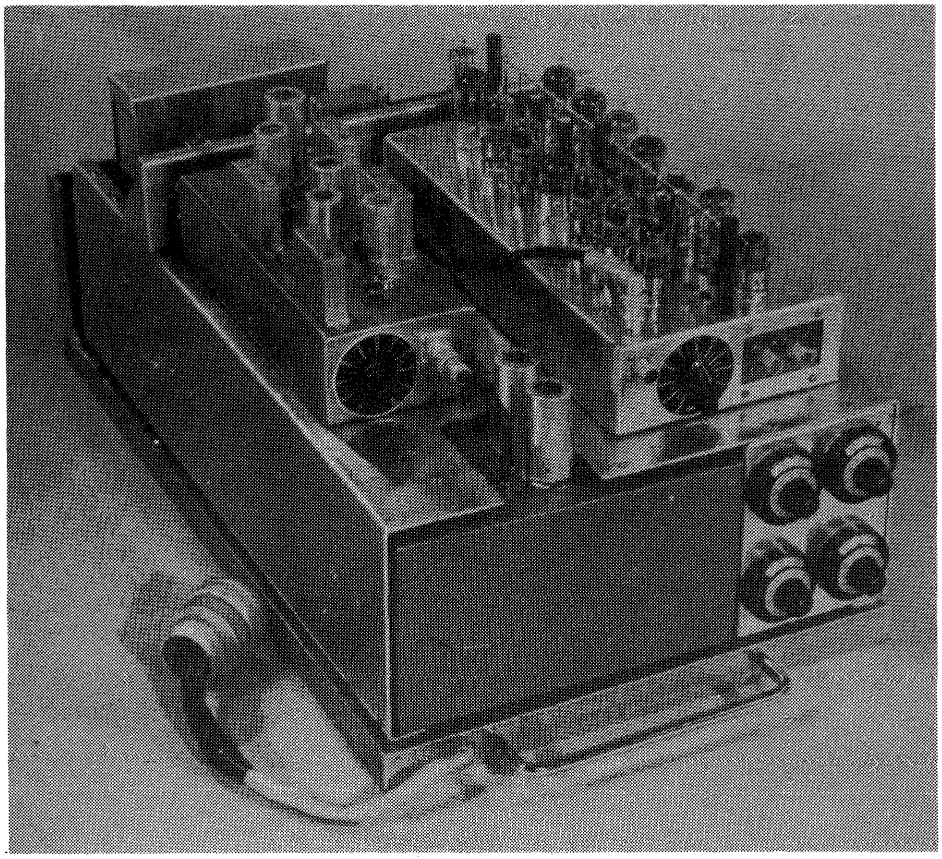


Figure 8—A complete plug-in unit

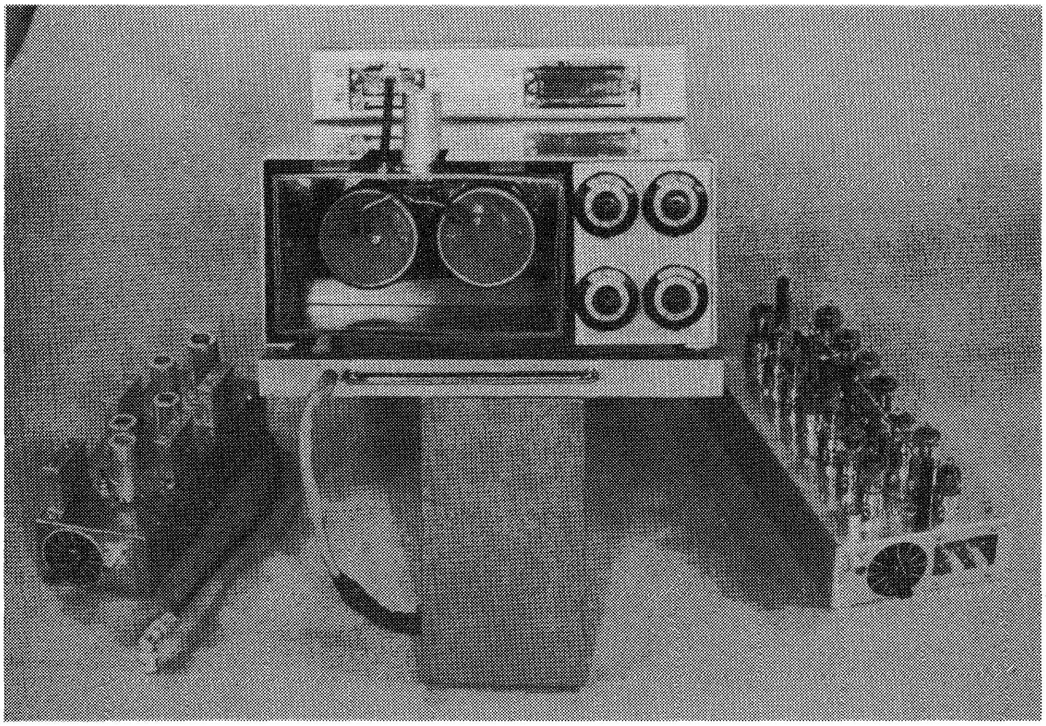


Figure 9—Plug-in subassemblies of the memory unit

ARTIFICIAL INTELLIGENCE

EWING L. LUSK, Track Chair
Argonne National Laboratory
Argonne, Illinois

Categorizing natural-language queries for intelligent responses

by KURT GODDEN
General Motors Research Laboratories
Warren, Michigan

ABSTRACT

A new classification scheme of natural-language queries based on a pragmatic notion called *satisfiability* is presented. A query is satisfiable if a direct response to it can be provided by the system. I claim that natural-language queries fall along a discrete satisfiable–unsatisfiable scale and that the production of informative English responses can be successfully organized around the satisfiability of queries. A system's semantic interpretation of a query can be shown to a user by a paraphrase in the response itself, and false user assumptions are correctable without disturbing the natural flow of the dialogue. These ideas have been implemented in a database dialogue system called DATALOG. Examples are presented using actual exchanges in the DATALOG system.

INTRODUCTION

The Problem of Literal Answers

A major difference between people and computers is that whereas computers usually respond literally to user requests and commands, people often, and perhaps usually, do not. Literal answers are often uncooperative and even misleading. In this paper I will discuss in general terms the question of providing intelligent responses to natural-language queries that range over a computer database of information. I provide a classification of query types based on the notion of "satisfiability" defined below. I discuss the desired response types to each kind of query and compare these idealized responses with actual responses produced by a working natural-language query system called DATALOG.[1]

Humanlike Responses

We may say that in general a friendly computer system that purports to understand at least some part of natural language should provide responses that are relatively humanlike. It should therefore respond with more than simple literal answers to English queries. Thus, a natural-language query system should give the user enough information in its responses not only to show that the user's query was properly understood, but also to indicate which parts, if any, of the user's request were not understood. Although many current query systems print a formal query interpretation or paraphrase and wait for verification before proceeding to answer it, quite a different approach is described here.

Grice's Cooperative Principle

In a classic paper, H. P. Grice[2] first stated what is known as the cooperative principle (CP), which can be summarized in the statement, "People tend to cooperate when speaking to one another." Grice goes on to categorize the CP into four classes, each of which has at least one maxim. These classes are the now familiar Quantity, Quality, Relation, and Manner.

Relating the CP to Database Queries

In a database query system one violation of the CP, giving too much information, would be a query such as, "List the female employees in physics who have PhDs" (of an employee database), to which a system might respond by listing an entire database of 1,500 employees. The requested information is present in the listing, but the response can hardly be called intelligent.

In a natural-language query system the problem is more likely to be that of providing too little information. This could occur, for example, by a system responding "None" to the query, "Which cars were the worst sellers in Smalltown in 1984?" Notice that this literal response is, in a sense, correct when all cars sold equally well or no cars were sold at all, and even when the question itself is inappropriate—when there is no dealership in Smalltown. Clearly, what we need is a system that responds more like an intelligent cooperating human across a wide variety of linguistic types of queries, each of which may have its own particular requirements for intelligent responses.

We may distinguish three broad linguistic types of queries that the DATALOG system currently responds to—imperatives, questions answered by *yes* or *no,* and *wh*— questions excluding *why*. In responding to any user query, whether or not that query is even understood, DATALOG should give an intelligent response. What follows is both a general discussion of intelligent responses and an explanation of DATALOG's approach to providing such answers.

CATEGORIZATION OF QUERIES

Satisfiability

Just as there is a syntactic-semantic continuum of grammaticalness (cf. Chomsky[3]), there is also a similar pragmatic continuum of query *satisfiability*. By satisfiability I mean the possibility, degree, and ease of answering a query. This informal definition may be clarified by an example. Consider a database of company employees. Both the queries "List the men older than 98" and "Show me the aardvarks in the math department" may not be answerable (no listable items), but for vastly different reasons. Further, I believe no one would argue the fact that with regard to the relevant database, the first query is in some sense more answerable than the second.

I now present new categories of queries based on this pragmatically motivated notion of satisfiability. Obviously, these categories hardly exhaust the query types in the English language and are intended as an initial classification only.

Satisfiable Queries

Within the DATALOG system we may distinguish three fully satisfiable query types corresponding to the three linguistic types of queries noted earlier. Thus for questions where the answer may be determined to be either *yes* or *no* (on the basis of the information provided in the query itself along with the information found in the database being used), such a question is satisfiable. Whereas a person often responds with a simple "yes" or "no" to such a question, a computer system should elaborate after this one-word response so that the user can see the system's interpretation of the input request. In other words, a computer system should provide a paraphrase of the user's question *in its response.* I emphasize that the paraphrase should be integrated into the response rather than displayed before answering the query, for the reasons discussed below.

A second kind of fully answerable query corresponds to *wh—* questions where the desired information is readily available in the database.

The third type of fully satisfiable query is composed of commands to display data items ("List the women with PhDs"), where there are indeed database items that meet all the restrictions imposed by the input command. All query types might also ask for statistical information on data such as averages, maximums, minimums, totals, and percentages. Again, such requests for statistics are fully satisfiable if there are items for which the statistics can be computed.

Partially Satisfiable Queries

Partially satisfiable queries are those to which a response can be supplied, but to which that response is not precisely of the nature expected by the user. Suppose a user tells the system to "List the tallest man," when in fact there are two men taller than all others and both are 75 inches tall. In the strictest sense, the computer cannot list the single tallest man, since there is not one such man. However, by the Cooperative Principle, it would certainly be expected of a person or a computer system to indicate to the questioner that there were two men who qualify as the tallest. I refer to this and other similar situations as a *superlative tie,* and is the first kind of partially satisfiable query.

A second partially satisfiable query type involves inaccurately specified numeric quantifiers, as in "List the five women in physics," when there are in fact fewer (no more) than five women in the physics department. Certainly in such cases, a necessary but not sufficient response will include the information that, contrary to the qustioner's assumptions, a different number of entities actually exists in the relevant group.

The third kind of partially satisfiable query considered here is one I refer to as *conditionally true.* By a conditionally true query I mean one whose answer *may* be false, but one for which falseness cannot be determined, since some of the relevant data are missing and the data that are known support the truth of the query. In such a situation, a person could reasonably expect a friendly response to say something to the effect that as far as can be determined, the query is true, but some of the relevant data are missing and if known could possibly falsify the query.

Unsatisfiable Queries

Either unsatisfiable queries cannot be answered at all with regard to what the query really concerns, or else an unexpected response can be given that differs in kind from the unexpected responses to partially satisfiable queries.

The first of the five unsatisfiable query types discussed here involves a too-strong restriction. An example is "List any men over 85 years old," when in fact no one, man or woman, is over 85. The typical response of a literal-minded computer would be to say "no data found" or to give no response at all.

The second kind of unsatisfiable query involves the notion of questioner assumptions. Suppose we consider the same query just cited but replacing the word *any* with the word *the,* "List the men over 85 years old." People using definite descriptions in utterances normally presuppose the existence of at least one entity satisfying the descriptions. The problem to be solved in a database query system is how to respond to queries that express false presuppositions. An automated query system should also inform the user that his or her presuppositions are invalid (although not exactly in those words). This is necessary because the user might otherwise pose another query with the same presupposition and thus waste time and perhaps fall into a frustrating loop of invalid queries and unhelpful responses. Kaplan[4] has called this type of behavior *stonewalling.*

The third kind of unsatisfiable query is one that differs in extent from partially satisfiable queries that are conditionally true but involve missing data. In the related unsatisfiable queries, all of the relevant data are missing, and therefore the system cannot make any statement directly related to the propositional content of the query.

The last two types of unsatisfiable queries differ from the previous three in that up to this point we have considered only queries that are successfully parsed. We can distinguish two types of unparsable input for our purposes here. The first includes queries whose individual words and phrases are in the system's dictionary, but which the system's overall grammar is unable to process. In such a case, the least complicated but nevertheless appropriate response is simply to tell the questioner that his or her query is not understood by the system and that perhaps the query could be rephrased. The second kind of unparsable input, and the last type of unsatisfiable query, occurs when not all the words or phrases in the query are recognizable. A simple approach may be to identify the unknown words and ask the questioner to double-check spelling or replace an unknown word or phrase with one that is known. A better approach is to first apply some spelling correction algorithms to such words.

This completes the categorization of queries by their satisfiability. There are of course other types of queries I have not discussed that could be added to this categorization. The problems of anaphora and ambiguity can be viewed in this framework of satisfiability as well. That is, when it is determined that anaphora or ambiguity is present, then the reference or reading is unsatisfiable until that reference or reading is resolved. This unsatisfiability may or may not be temporary and needs to be reevaluated after resolving the anaphora or ambiguity. If this resolution cannot be accomplished, then we have a new category of unsatisfiability, which has its own requirements for a satisfactory response. Taking a lead from humanlike reactions to unresolved anaphora or ambiguity, the system may simply ask for clarification ("Who does *he* refer to," or "What do you mean by. . . ?").

RESPONSES IN THE DATALOG ENGLISH QUERY SYSTEM

Choosing Responses

In this section I wish to briefly review the responses generated by DATALOG with respect to *satisfiability* to see how DATALOG's responses address this notion.

Responses to Satisfiable Queries

Answers to fully satisfiable questions requiring yes/no answers provide query paraphrases as part of the answer. This ensures that the user can verify that his or her *intended* query is being answered in case it is ambiguous. For example, the query "Are any women older than the oldest man in math?" may be interpreted as asking if any women are older than some man, or if any of a certain group of women are in math. Just which interpretation is being answered is indicated in the response, "Yes, there is one employee whose sex is female and whose age > 52." When attribute values must be computed, they are substituted into the response (like the *52* above).

Responses to *wh—* questions may appear as lists of database items if a list is essentially what is being asked for, as in "Which men are in the math department?" Whereas a person would answer a question such as "How old is Smith?" with a number or a short phrase, the response in DATALOG is again a list showing the item in question along with any default attributes to display in addition to the attribute actually requested. Other *wh—* questions elicit an English response from the system. "What % of the men earn more than the woman with the greatest salary?" receives the response "For 3.7% of the employees where sex is male it is true that salary > 55000."

Commands can also result in displayed lists of items. If a user tells the system to "List the men with a height greater than 70 inches," it will show the explicitly mentioned height attribute, the sex attribute which is implicit in the reference to men, and also the default name and department attributes.

Responses to Partially Satisfiable Queries

There are three varieties of partially satisfiable queries, as discussed above. The first involves superlative ties. The system conveys this information to the user by noting the discrepancy in the response:

USER: Is the shortest man shorter than 65 inches?
COMPUTER: Yes, of the 3 employees where sex is male and height is least it is true that height < 65.

Here the user can note that there must be three men all with the same height who are shorter than all other men. If only one or two of those three were shorter than 65 inches, the response would have indicated just that with a sentence such as "...it is true for 1 of the 3 that height < 65."

A slightly different approach is taken with a specific but inaccurate numeric quantifier:

USER: List the 7 women in physics.
COMPUTER: The number of employees you are asking about is equal to: 5 < followed by a list of those 5 >

The response to a conditionally true query tells the questioner that to the best of the system's knowledge, the query is true:

USER: Are all the math employees older than 24?
COMPUTER: Yes, for all 8 of the employees where department is math it is true that age > 24 except for those employees whose data is incomplete.

Unsatisfiable Queries

When the user poses a query with a too-strong restriction, the system provides a specific message indicating the nature of the unsatisfiability:

USER: Are there any men with master's degrees who are over 85 years old?
COMPUTER: I cannot find any employees where age > 85.

Notice the response is not that there are no men with master's degrees older than 85; rather, a more specific response is given to indicate that no one in the database is that old. Similarly, when an invalid user assumption is detected, that failure of assumption is pointed out:

USER: Which of the level 7 employees who are over 49 years old earn more than $45,000?
COMPUTER: For the employees where level = 7 there are none where age > 49.

This response avoids the kind of stonewalling behavior Kaplan pointed out, which could occur if the response had been "None."

The next kind of unsatisfiable query involves missing data that trigger a response indicating the system's inability to respond to the content of the query:

USER: Is Green older than 40?
COMPUTER: I do not know the age of the employee called Green.

In a situation where the user may use words that all appear in the system lexicon, but which are used in a novel or ungrammatical construction, DATALOG suggests that the user rephrase the query and try again.

Finally, another kind of unparsable input is due to unknown words or phrases that appear in the input and that may or may not be correctable. Upon encountering an unknown word, DATALOG displays a message asking for a replacement word or phrase, or else an indication to delete either the unknown word or the whole query.

Computing the Responses

The reason DATALOG is able to compute these kinds of intelligent responses is that its architecture differs from the more traditional architecture of natural-language query systems. In most other systems, the input query is parsed and translated to a formal query in an existing database language. This formal query is then shipped off to the underlying database management system, where it is applied and the resulting output is displayed to the user. This output is only slightly modified, if at all. The result is that short, literal responses are the norm. The problems inherent in such an approach have already been discussed.

DATALOG's architecture differs in the formal query representation and also in the processing of that representation. In DATALOG, an English query is parsed and a formal representation of its semantics is constructed. This representation, however, is not the language of an existing database system. Rather, it is a custom-designed general frame-structured representation system.[1] Furthermore, this structure is not "executed" to obtain the system's response. Instead, there is an additional module that *interprets* this frame structure and accesses the database directly through generic access functions. There is no limitation to the number of times the database can be accessed. Access is performed as required by the semantics of the input query. The information or data retrieved from the database are then either displayed to the user or *incorporated back into the frame structure.* When natural-language responses are called for, the resulting frame structure itself forms the building blocks from which to construct the answer. In this way, indirect references to values as contained in the user query are often seen as those direct values in the response (examples above). Thus, this extra response module provides great flexibility and power for the generation of intelligent answers.

RELATIONS WITH OTHER RESEARCH ON RESPONSES

Any work purporting to provide something new in the way of cooperative responses in natural-language query systems must in some sense measure itself against the pioneering work presented by Kaplan.[4] In this section I will point out some of the advances in this area that DATALOG exhibits and present those advances in the context of Kaplan's COOP system.

Certain features are present in both systems, such as indirect reference to database entries by description rather than by name and correction of false user assumptions when detected. This latter capability, one of the major features discussed by Kaplan, is divided into two categories by him. The first is simply telling the user that certain presumptions (Kaplan's term) indicated by the query are not valid. This feature is presented above under the notion of false *assumption.* The term *assumption* as I have used it here is equivalent to Kaplan's use of *presumption.* The other corrective type of response in COOP Kaplan calls *suggestive.* It is triggered when a query cannot be directly answered but certain database information is presented under the belief that it may be relevant to the user's original query. DATALOG also provides this capability in various different ways. In the DATALOG system, however, database items thought to be relevant are available not just when a query fails, but also when a query is successfully answered.

There are other extensions to the cooperation provided by DATALOG. Unlike COOP, which is intentionally limited to *wh—* questions and data display, DATALOG is intended eventually to be a production system and therefore has a fairly broad class of syntactic constructions within its grammar. Each of these brings its own demands for informative responses, as presented in earlier sections.

COOP and INTELLECT[5] both present the user with a paraphrase of the system's interpretation of the input query. This paraphrase must be confirmed by the user before the system proceeds to respond to that interpretation. While this ensures the system interprets the query as the user intended, it can be quite annoying. INTELLECT permits the user to inhibit this facility, but then the user may unknowingly receive a wrong or misleading response if an incorrect interpretation is the one followed by the system. Since DATALOG is designed to be humanlike in its responses (when that is deemed appropriate), a different approach is taken. This approach is that previously mentioned of providing the paraphrase *within the query's answer.*

A particularly revealing example of how the paraphrase method employed in DATALOG is of value involves queries with disjoined subjects, "Is Drake or Smith taller than Jones?" In spoken English this sentence has different interpretations depending on the intonation of the utterance. With rising intonation there is no speaker assumption about either person being taller than Jones, whereas one is assumed taller than Jones when spoken with falling intonation. The user could be vocalizing the query in either manner while typing a single request. DATALOG's response indicates clearly what the user needs to know through its paraphrase contained in the answer, "Yes, for the employee called Drake it is true that height > 71." If neither person had been taller than Jones, the response would similarly have been quite clear, "No, it is not true for the employees called Drake or Smith that height > 71." The point here is that no matter what the user intended to mean, the information desired is present in the response and there is no need to stop and ask for confirmation of that interpretation.

The ability to interpret pronouns and intelligently indicate to the user exactly what the referent is is the subject of current work on DATALOG. This will be the topic of a future report.

CONCLUSIONS AND IMPLICATIONS FOR NATURAL-LANGUAGE RESEARCH

In the preceding sections I have presented an initial classification of natural-language queries based on the notion of "satisfiability." These notions have been successfully implemented in a working natural-language system called DATALOG. Initial observations indicate that the resulting responses are more humanlike in that they provide English answers when appropriate, do not require confirmation of interpretation and yet avoid possible misunderstanding, and conform generally to the Cooperative Principle.

Currently, DATALOG uses canned responses constructed from English fragments and modified segments of the system's formal interpretation. As development continues and the grammar's scope is expanded, the code that constructs these canned responses will grow proportionately. The result will be an expansive response module that could become increasingly difficult to maintain. A possible solution to this problem would be to construct a *generative* grammar to dynamically construct English responses as needed. There is some research in the area of language generation.[6] I suggest that such a grammar may be fruitfully developed that uses satisfiability as its organizing principle, rather than purely syntactic or semantic constructs. This is then a call for research into pragmatic grammars for language generation. The groundwork for such a grammar could be laid by continuing to identify and refine the categories of pragmatic satisfiability.

REFERENCES

1. Hafner, C. D., and K. S. Godden. "Design of Natural Language Interfaces: a Case Study." Research Publication GMR-4567. Warren, Michigan: General Motors Research Laboratories, 1984.
2. Grice, H. P. "Logic and Conversation." In P. Cole and J. L. Morgan (eds.), *Syntax and Semantics: Speech Acts* (Vol. 3). New York: Academic Press, 1975.
3. Chomsky, N. *Aspects of the Theory of Syntax*. Cambridge, Mass.: The MIT Press, 1965.
4. Kaplan, S. J. "Cooperative Responses from a Portable Natural Language Query System." *Artificial Intelligence,* 19 (1982), pp. 165–187.
5. INTELLECT Query System User's Guide (2nd ed.). Newton Centre, Mass.: Artificial Intelligence Corporation, 1980.
6. Mann, W. "Text Generation." *American Journal of Computational Linguistics,* 8 (1982), pp. 62–69.

How to get a large natural-language system into a personal computer

by BOZENA HENISZ THOMPSON and FREDERICK B. THOMPSON
California Institute of Technology
Pasadena, California

ABSTRACT

The answer to the question of how to get a large natural-language system into a personal computer lies in the paging architecture of the system. The key is to use the input sentence, in conjunction with the lexicon and grammar table, to identify the minimal segments of both object code and data that must be brought into main memory. Once such a maximally paged architecture has been effectively implemented, it has wide ranging implications for process integration, networking and knowledge base distribution, and for the software engineering environment. The Natural Access System optimizes this architecture and exploits these implications.

The Natural Access System is a large natural-language system. It is now implemented and running on a personal computer (PC). This paper tells how we were able to get such a large system on such a little computer.

THE OBJECTIVES OF NATURAL ACCESS

The Natural Access System has evolved over a number of years with the object of providing truly natural access to the computer for intelligent people who may not be programmers or even computer literate. Because of the academic setting of our research, we have been able to take a fresh look at this problem, and to guide our research through considerable experimentation.[1–3] We have assumed that most people will have direct access to computers in civil and business organizations, in research labs, on engineering floors, among management staffs, and in their homes. The forms and facilities of this access are still evolving in response to the question, what is the proper form for this access and what are the requirements for a computer software system that will provide natural access to computers?

Experience with existing systems makes it abundantly clear that almost every application of computers is a special application not adequately served by any single general purpose system, and that the requirements of any specific application are constantly in flux, thereby necessitating constant updating and extending of any system implemented for its support. Yet, it is not economically viable to program a new system from scratch for each application and to constantly retrofit that detailed special purpose design. What is more tenable is a system design that serves a defined range of applications, and within this range, is readily adaptable to each specific application; a system which in its design supports its own customizing and recustomizing to specific requirements. Today, the computer software industry is structured to respond to these realities as evidenced by the proliferation of software firms that specialize in narrow applications areas, and others in the development of sophisticated tools applicable to a variety of uses. A general system must accommodate itself to this reality. It must also attend to the problem of software engineering. The design of the Natural Access System has been undertaken in light of these considerations.[4,5]

The domain of applications for which the Natural Access System is designed can be roughly described as those where

1. the individual's interaction with the personal computer dominates the computing task;
2. the tasks are of a reasonable degree of complexity requiring more than simple record keeping; and
3. an individual's interaction with the computer impinges on other people. Thus the computer acts as a medium for communication.

This domain specifically excludes *compute-bound* applications such as those arising in the physical sciences, and applications such as airline reservations systems where the capability to handle large volumes of simple transactions is required. Applications typical of the domain we intend to cover are local area networks serving business organizations, computer support that provides intelligent interfaces to the experimental apparatus of research teams, the media for computer-aided design and the coordination of design in an engineering shop, and the household computer with links to banks, stores and wide-ranging sources of information.

In order for a computer system to be responsive to the needs of an organization, it must contain a great deal of "knowledge" about the domains with which the organization is concerned. Knowledge can take the form of databases whereby the structural linkages establish relationships among database entities, procedures that express implicit meanings not discernible in structure alone, and assertions wherein implications are developed through symbolic manipulation. A "knowledgeable" computer must be able to integrate these forms with the understanding of succinct instructions; an understanding that prompts it to marshal data, processes and assertions in complex ways that respond to the immediate needs of the user.

In many organizational environments, the knowledge base is rapidly changing. Its maintenance becomes one of the major activities of the organization itself. This is true of corporate information systems, systems for the coordination of public agencies, and the many systems that support management, engineering and military staffs. Maintenance of the knowledge base is not a single level of activity since managers, clerks, engineers and applications programmers will all be actively interacting with the knowledge base. The knowledge base will indeed be their integrated, dynamic group memory; the information context that gives substance and meaning to their otherwise diverse activities. The analyses of the dynamics of these varied interactions must be a ubiquitous aspect of the design of such a system.

In a complex organization of professionals such as the corporation, research laboratory, or military staff, there can be no single knowledge base, but many, each with its own rate of change, content and responsible agents. On the other hand, these various knowledge bases are by no means independent. Figure 1 illustrates the relationships between knowledge bases that may exist in a simple organizational structure.

In a large organization such as a multi-national company,

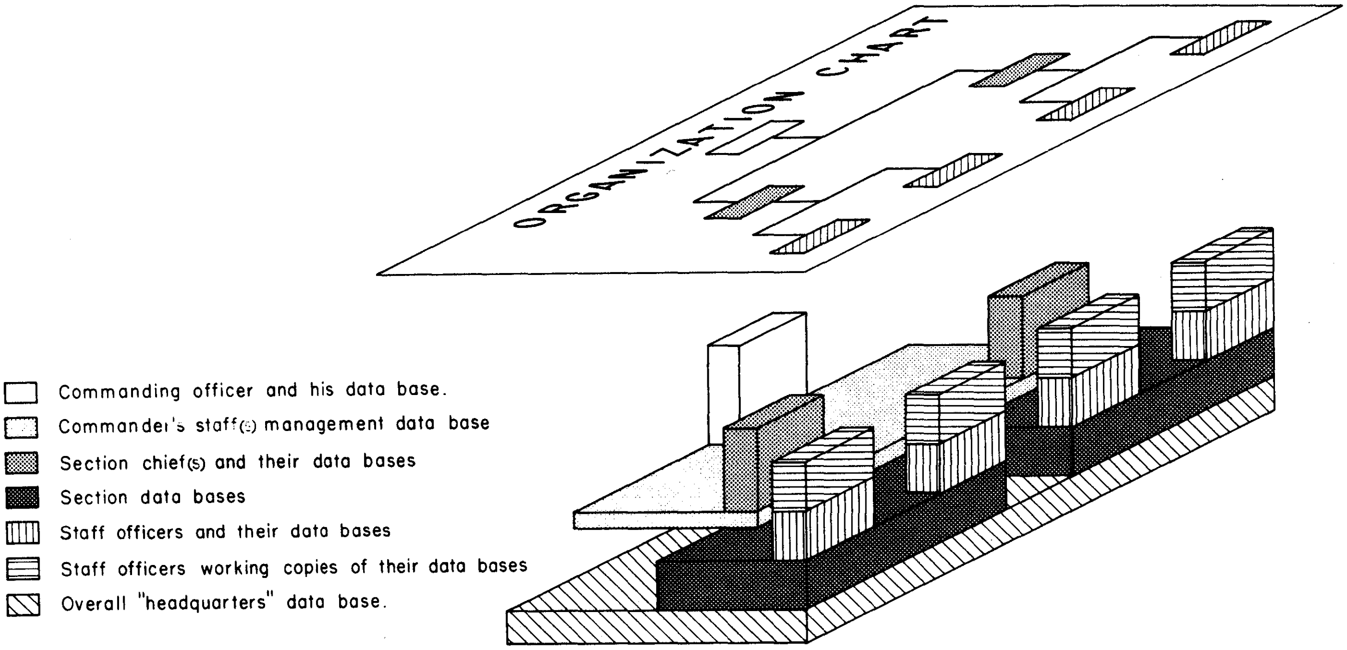

ORGANIZATION CHART
□ Commanding officer and his data base.
▨ Commander's staff(s) management data base
▨ Section chief(s) and their data bases
■ Section data bases
▥ Staff officers and their data bases
▤ Staff officers working copies of their data bases
▧ Overall "headquarters" data base.

Figure 1—Relationships between contexts in a small organization

the intensity of inter-knowledge base activities also has spatial dynamics, because there is less communication between distant corporate entities. The nature of interactions between offices that are vertically related (i.e., one being organizationally subordinate to the other) is different than that between two laterally related sister offices. Therefore, a system for such organizations must be able to support many knowledge bases and incorporate means for their interaction.

What should the form of communication be between a member of the organization and the system? There are many forms for human–computer interface under development. However, the context of the professional's interaction will largely determine the appropriate form; designing a piece of machinery or developing the load plan for a ship clearly calls for graphic interaction; standard queries of a repetitive nature call for the use of tableaux or forms; when the computer is required to respond through a complex decision structure to relatively few but varied inputs, a system-guided dialogue is desirable; in a word processing environment, simple single-key instructions with cursor control are called for. All of these appropriate interfaces should be available for users to invoke as a natural aspect of their own structuring and development of various knowledge bases. Further, the user should be able to employ various forms of interface to the same underlying knowledge base, fully integrating graphics, images, text and dialogue within that knowledge base as deemed appropriate, when all concern the same domain of data and process.

These two requirements, the ability of professionals to arrange their own forms of interface, and the integration of available interfaces with each other and with the knowledge base, suggest a single form of interface, prior to the invocation of the others, which is sufficiently flexible, expressive, and natural to be used both as an interface form in itself, and in the design and specification of other interface forms. In a system that gives truly natural access, this prior interface should be a simple dialect of English. English then becomes the primary language by which the user can customize his* forms of interaction, and to which he can fall back when the more succinct forms he has evolved are no longer sufficient in light of changing needs.

THE NATURE OF CONTEXTS

We will give more specific structure to this general notion of knowledge base, referring to it as a *context.* Roughly speaking, a context encompasses a vocabulary, language, definitions, database and possibly special procedures and extensions. A professional at the console will usually be interacting with the computer in such a context. Each professional will probably have a number of contexts; perhaps one encompassing administrative matters for keeping work logs, personal budgets and schedules, files of memos concerning assignments, and progress reports; one containing the specifications, design ideas and requirements for each of the projects in which he or she is participating; and possibly a bibliography. At any given moment, one may have made a copy of a working context, and be trying out some new ideas using the copy so that the original is not irretrievably destroyed.

The idea of a context can more easily be grasped by considering how such a context can be created in a Natural Access System. Initially, the system contains just one context, called the *BASE Context.* BASE contains a simple dialect of English

* The masculine pronominal forms are used in this paper simply because of the brevity of *he.*

as its primary language, a math and statistical package, text, image and graphic packages, means to add vocabulary and definitions for adding and changing data, and other ingredients of a fully-implemented knowledge-based working environment. The vocabulary of the BASE context contains all of the "small" words (i.e., *and, what, was, exceeds,* etc.), terms such as *square root* and *average,* and commands such as *create, list,* and *display.* To create a new context, the professional enters the command:

⟩Base . . . on ---

Suppose his organization, called *Corporate,* has a general context containing addresses and phone numbers, cost-center rosters and task assignments, and further suppose that he wishes to have this data at his fingertips, and to add to it his own administrative records for his private use. He may then type

⟩Base my admin on Corporate
⟩Enter my admin

and proceed to use this new context for his own purposes. Referring back to Figure 1, when one context rests on another, this basing relationship is indicated. If Context B is based upon Context A, any changes in A will automatically and immediately be reflected in B; however, changes in B will not affect A in any way. One context can also be based upon many contexts, thereby allowing it access to all of their knowledge bases. Many contexts may also be based on a single given context.

THE LANGUAGE PROCESSOR OF THE NATURAL ACCESS SYSTEM

A central aspect of the concept of a natural access system is that of the naturalness and flexibility of the user language. If the user is indeed able to use his natural language, the language processor for that language is never far from system architectural concerns. We first examine the nature of that processor.

The language processor of the Natural Access System is a simple syntax directed interpreter. The system is based on *compositional semantics* and it has *procedural semantics.* Semantics is that part of the specification of a language that deals with meaning. Compositional semantics is a way to interrelate the meaning of a sentence to the syntactic structure of the sentence. Compositional semantics assumes that the syntax of the language is given in the form of general rewrite rules, each associated with a semantic interpretation procedure. Applying the rules to the syntactic analysis of the sentence, a procedure called *parsing,* results in a parse graph. The semantic procedures associated with the applicable grammar rules are composed or *compiled* using the parse graph, and the result evaluated. Careful exposition of this concept of compositional semantics shows that the role of part of speech in rules of syntax is to guarantee that the arguments of the associated semantic procedures are of the correct type (in the sense of "type" in such programming languages as Pascal). For example, consider the following rules (where "num" stands for "number", "att" for "attribute" and "comp" for "comparator")

R1: ⟨num⟩⇒⟨num att noun⟩ "of" ⟨class noun⟩
R2: ⟨num⟩⇒⟨article⟩ " " ⟨num⟩
R3: ⟨num⟩⇒⟨num⟩ " + " ⟨num⟩
R4: ⟨clause⟩⇒⟨copula⟩ " " ⟨num⟩ " " ⟨comp⟩ " " ⟨num⟩
R5: ⟨sentence⟩⇒⟨clause⟩ "?"

and the sentence

Is 40 greater than the number of students + 5?

We assume that a preprocessor, referring to the lexicon, has parsed this sentence as follows:

```
⟨copula⟩ ⟨num⟩ ⟨comp⟩......⟨article⟩ ⟨num att noun⟩
Is        40    greater than  the       number
  ⟨class noun⟩ ⟨num⟩
of students    +5   ?
```

The above rules result in a *parse tree* for this sentence, shown in Figure 2.

In compositional semantics, a semantic procedure is associated with each rule of grammar. For example, consider the rule

R3: ⟨num⟩⇒⟨num⟩ " + " ⟨num⟩

with the associated semantic procedure PLUS PROC. Clearly, PLUS PROC adds the two constituent numbers. By this grammar rule, we have the parsing

```
⟨num⟩.......
⟨num⟩ ⟨num⟩
3     + 5
```

and thus the meaning of "3 + 5" is

PLUS_PROC(3, 5) = 8.

Now associate with each of the above five rules the following procedures.

```
⟨sentence⟩..........................................................................
⟨clause⟩............................................................................
                                      ⟨num⟩ ....................................
                                              ⟨num⟩ ..........................
                                              ⟨num⟩ ..........................
⟨copula⟩ ⟨num⟩ ⟨comp⟩............ ⟨article⟩ ⟨num att noun⟩   ⟨class noun⟩   ⟨num⟩
Is        40    greater than          the        number         of students     +5     ?
```

Figure 2—Parse tree for the sentence

R1: ⟨num⟩⇒⟨num att noun⟩ "of" ⟨class noun⟩	: FIND_ATT
R2: ⟨num⟩⇒⟨article⟩ " " ⟨num⟩	: NO_OP
R3: ⟨num⟩⇒⟨num⟩ " + " ⟨num⟩	: PLUS_PROC
R4: ⟨clause⟩⇒⟨copula⟩ " " ⟨num⟩ " " ⟨comp⟩ " " ⟨num⟩	: COMP_PROC
R5: ⟨sentence⟩⇒⟨clause⟩ "?"	: OUT_PROC

It is not surprising to find that the semantic procedure associated with some rules is the NO OP procedure; such rules govern *function* words that play a purely syntactical role (*the* in some linguistic contexts may not be a function word). Referring to the above parse tree, we see that the meaning of the example sentence is given by the functional composition

OUT_PROC(COMP_PROC([40], PLUS PROC
(FIND_ATT([number], [students])), [5])))

Note that we have associated the FIND_ATT procedure with the rule:

R1: ⟨num⟩⇒⟨num att noun⟩ "of" ⟨class noun⟩ : FIND_ATT

The meaning of the word *number* is given by means of the procedure that counts a class and returns the number of elements in the class. A pointer to this procedure is found in the lexicon as part of the definition of *number*.

Rule R1 would also apply to the phrase

```
⟨num⟩ ........................................
⟨num att noun⟩ ...................... ⟨class noun⟩
⟨unary operator⟩ ⟨num att noun⟩        ⟨class noun⟩
average           age                 of students
```

In this case, the first constituent to which R1 applies is not a simple procedure as in the above example, but a more complex structure whereby *age* is most likely given by a table; FIND_ATT uses this table to find the set of numbers which are the ages of students. FIND_ATT then examines the unary operator *average,* and finding it to be given by a procedure, applies this procedure to this set of numbers.

Thus we see that the meaning of a word or phrase can be given by a pointer into the database by means of a procedure, or even a complex structure of database entries and procedures. For this reason, *compositional semantics,* when encompassing this possibility, is also referred to as *procedural semantics.* In discussing programming languages such as Simula, where the meaning of semantic entities can have a mixture of structural and procedural aspects, the term *object-oriented language* has been used.

The language processor of the Natural Access System is a general rewrite rule–procedural semantics processor. It is general in that the syntax of any well defined language can be specified by a general rewrite rule grammar. On the other hand, procedural semantics is not sufficient to define the semantics of useful dialects of natural language because the understanding of a natural language utterance depends on the context of the dialogue in which the utterance occurs. Methods for incorporating this dependence on dialogue context into a natural access system constitute one of the most important and active areas of artificial intelligence research. The concepts of Frames (introduced by Minsky),[6] Scripts and Mops (introduced by Schank),[7] and Partitioned Networks (introduced by Hendrix)[8] are significant contributions to this area of research. The Natural Access System falls short of incorporating methods as far reaching as these three. However, in the handling of anaphora and case frames of verbs, it goes well beyond procedural semantics.[9] It does so by explicitly handling these aspects in procedures that are called in the language processing module but independently and in parallel to the central parsing and semantic processing procedures.

The Natural Access System (NAS) handles many languages, both dialects of natural languages such as NAS English and NAS French,[10] and programming languages used in the NAS Metalanguage environment. One of the central architectural features of the Natural Access System is that all of these cases use the same language processor. Therefore, all NAS languages are syntax rule-based and procedural, and may be object-oriented in the above sense. They differ only in their syntax rules and associated semantic processing procedures. Thus a language is implemented in the Natural Access System by declaring its syntax (any general rewrite rule grammar), and for each syntax rule, the associated semantic procedure. This has great advantages as described below.

The Natural Access System is sentence-driven. One can think of the Natural Access System as operating in the following paradigm.

1. The system types a ")", indicating that it is ready for user input, and waits for the user to respond.
2. The user enters a sentence, or more generally any string ending in an *end of entry* key. (We will often refer to such a string as a *sentence* even when it does not parse to ⟨sentence⟩.)
3. The system responds by processing the sentence, displaying the results on the user's terminal, and cycling back to the first step.

Many aspects of Natural Access System processing are organized around this paradigm. Here are some typical examples.

1. During the processing of a sentence, the processing procedures may make use of various forms of temporary work space; at the end of sentence processing, these are returned to the general work space pools.
2. If the user input is not a true *sentence,* the system tries a variety of corrective procedures (i.e., spelling correction), assuming that the input by itself was supposed to constitute a meaningful expression.
3. The user might inadvertently enter a string which, for one reason or another, takes an inordinate amount of time to process. In this case, the user can at any time press the BREAK key and abort the current processing. In this circumstance, the system "cleans up" its working environment, readjusts itself, and is ready to receive the next sentence.

CONSIDERATIONS IN THE DESIGN OF THE CENTRAL ARCHITECTURE

Central to the design architecture of the Natural Access System, as with any "next generation" system, is the difficulty such a system would have were it to function within one hardware level of memory. The discrepancy between *main memory* and *peripheral memory* will continue to be significant, and must be a consideration in the management of the object code and knowledge base.

Because the total body of just the object code involved in a system such as the Natural Access System is too large to fit in main memory at one time, it is tempting to nibble away at integration through the expedient of increasing main memory. But dependence on this stopgap method impedes progress toward a rational solution of the problem. The object code of a truly integrated system includes not only the operating and language processing subsystems, but the semantic procedures of the primary languages, text processor, image processor, database management system, display management procedures, user language-specific semantic procedures, and the many application-specific procedures developed by programmers for their users' domains. The integrity of the user context depends on the unlimited capacity of the code body to accommodate extensions as the user community matures and the size and complexity of the computer's tasks grow. Segmentation of this code thus becomes a major architectural decision.

The solution to this problem for current generation systems is epitomized by the UNIX system. The resident UNIX code is small. A UNIX system environment has a number of major code bodies (*shells*) which can each be loaded into main memory and given control and which communicate through a common filing system. These shells are typically large enough to embody a significant form of user interface.

Typical shells such as word processors, database management systems, or application programs that handle prescribed yet versatile domains of user considerations can be characterized as *process oriented.* If the user is a system programmer, process orientation fits the conceptual organization of his task well. He simply writes and edits code, compiles it, and applies it to a specific domain. His interests are often with new systems having small programming infrastructures and programming languages that are quite free of domain-specific utilities. There is no need in his world to mix media; source code is in a fixed medium, namely text (though it may concern the manipulation of several media).

Indeed, the UNIX architecture serves computer research and development laboratories well. However, segmentation based on process orientation does not fit the conceptual organization of end users. Text, pictures, data, display formats, statistical processing, and distributed sources of data commingle in the user's conceptual world. Thus in the computer system context with which the user is engaged, such requests as

)Display a bar graph of the variation in net sales of each sales region over the last three months.

and

)How many originators of memos referring to ICOT were there in each organization?

require procedures from the display management package, statistical package, telecommunications package, database management package, and text processor, albeit from only a small fraction of the procedures in any one of these packages. For the user to exit from his word processor, enter his electronic mail system to retrieve the net sales of his various sales regions, invoke his statistical shell, and finally go to his display formatter to see the desired bar chart is not natural access. To hide these transitions from the user in yet another shell, but one that leaves him little flexibility and does not relieve the situation, is not a good solution, either.

Segmentation based on processing function is also not germane to the needs of the applications programmer. He does not program a new system from scratch, but extends the existing system capabilities for his user clientele. The application area he is building or extending, depending on what part of the application programming chain he is in, already has an extensive infrastructure of database processing and special display management utilities embodying a great deal of domain-specific knowledge. Much of his programming consists of calls to these utilities. In present practice, he has no choice of programming language. In a UNIX environment, he is extending an existing application shell, but unless he is willing to give up control of paging faults and use a virtual memory (giving up good response times by avoiding thrashing even though his knowledge of his particular application would allow him to do so anyway), he is limited in the extent of the application package he can put together by the capacity of main memory. Even if he chooses to use virtual memory, a choice often forced upon him by the available compiler for the language, and to pay the unknown price for thrashing when his application gets realistically large, he is faced with yet another dilemma; for when he wishes to use small parts of procedure packages already existing in other shells, he must either reprogram them, or use utilities that swap shells for him, making available both the useful procedure and its brothers, sisters, aunts and uncles who are "swapped along" for the ride.

In consideration of the context of the end user and that of the applicatons programmer, the Natural Access System uses a distinctly different segmentation architecture as presented next.

CENTRAL ARCHITECTURE OF THE NATURAL ACCESS SYSTEM

When a user of the Natural Access System enters a sentence, the entry is parsed. As a result of this parse, the relevant semantic procedures associated with the rules, and perhaps procedures associated with words in the sentence, are identified. Calls to these, and the utilities they may call, are compiled and executed in the processing of the sentence. With the exception of the small resident code, no other procedures are

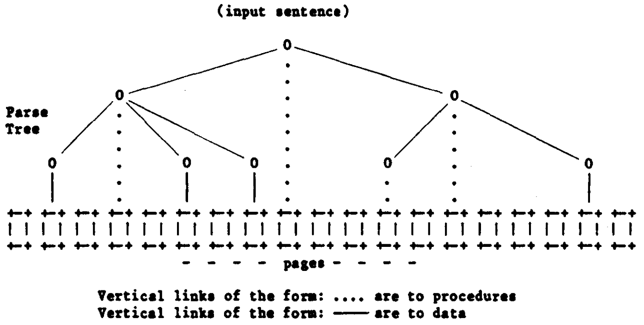

(input sentence)
Parse Tree
pages
Vertical links of the form: .... are to procedures
Vertical links of the form: —— are to data

Figure 3—Relationships between parsing and page loading

relevant, and no other procedures need be in main memory while the sentence is being processed (indeed, these may not all need to be in main memory at the same time). Access will presumably be made to database entities. Thus, the relevant database entities must also be brought into main memory, but these too constitute a very small part of the entire database.

In the Natural Access System, procedures and data entities are paged individually on the same peripheral memory pages, and into the same page-frame area in main memory. To this end, the Natural Access System is its own page manager. It can delegate control of the locking and marking of pages to both the language processor and the individual semantic procedures and utilities. For example, the grammar table is made up of a binary search tree and a rule definition part, all of course on pages. When the parser is called, it can load and lock the entire binary tree while individual syntax time procedures are brought in as needed in conjunction with selected rules. This drastically reduces parsing time otherwise required if parts of that tree were swapped with all syntax procedures as a package every time one of them were needed. Procedures can determine how many page frames are dynamically available to them and optimize their page loading and locking at the last moment by making use of information available about the size and disposition of the structures in their calling sequence as well as the processes they will be carrying out. Such information is not available to a system-level segment manager.

Figure 3 gives a schematic view of the relationship between parsing and page loading. In processing the input sentence, only the pages linked directly or indirectly to nodes in the parse tree need be brought into main memory. The number of pages available on peripheral memory is very large. Several hundred of those pages will contain semantic procedures; many others will contain specialized utilities such as database, graphic, screen management, text, and image processing utilities. Realistically, database data pages will run into the thousands. Knowledge bases will use pages containing complex structures; most of the page leaves will be page pointers to other structures. A given sentence will need to call upon only a small number of these.

Secondary design decisions that are crucial to the viability of this architecture involve

1. the paging algorithms;
2. the section of what system code is to be resident and what is to be paged;
3. the size of the page frame area of main memory; and
4. knowledgeable optimizing algorithms that are built into procedures for controlling page loading.

The test of this architecture is response time. Actual tests of the system indicate that the processing of a typical English query requires something in the order of 200 page loads. Average throughput response time on a Motorola 68010 chip-based computer for sentences such as

"List the destination and cargo-type of each shipment on the Tokyo Maru."

is four seconds; for the sentence

"What is the average population of cities?"

where there are perhaps 1500 cities, it is 20 seconds. These times reflect the following layout of main memory as required by the Natural Access System

1. resident NAS code	150K bytes
2. page frame area @ 1K	120K bytes
3. list processing area	100K bytes
4. miscellaneous work areas	30K bytes

The advantages of the NAS architecture is evident. All of the capabilities provided by the system are equally available. These capabilities can be extended in any direction by simply incrementally adding new syntax rules and their associated semantic procedures. The user's input sentence is all that is needed to orchestrate the loading of the data and procedures necessary for the processing of that sentence. The repertoire of capabilities available for that processing is not limited by the size of main memory, but rather by the effectively unlimited size of peripheral memory. Thus, the notion of integration, as so often discussed in personal computer literature, is achieved to its fullest extent.

NETWORKING

Once all procedures and data are paged on the same pages, and paging is managed by the Natural Access System, another opportunity becomes available. The page is an ideal packet for communication, and the page address is ideal for managing that communication. A Natural Access System installation can be viewed as a network of NAS stations, each consisting of a computer (it is appropriate to think of these as personal computers) and at least one hard disk with a minimum capacity of, say, 20 megabytes. We are presently using a page size of 1024 bytes. Thus, such a disk would hold 20,000 pages. We can also think of stations as grouped into clusters, each with a designated agent, and agents tied together in a wider net. (The cluster grouping allows stations to be added to or taken out of a cluster without reconfiguring the whole net.)

In the Natural Access System, a page pointer consists of a

1. cluster number (1 byte);
2. station number within a cluster (1 byte);
3. volume number relative to the station (1 byte);
4. block number on the volume (3 bytes); and
5. page offset on the page (2 bytes).

Thus, the bound on the amount of network information accessible to any station is the prodigious figure of 2^{48} pages or over 10^{17} bytes. Page pointers are ubiquitous throughout the data and system structures (e.g., grammar table, and database) of the Natural Access System. However, only the paging module "knows" anything about the structure of these pointers. When a system or semantic procedure requests a page, the paging module at the station checks for the page in main memory. If the page is not found, it checks for it on the station's own volumes. It may need to go to another station or, through its agent, to another cluster for the page. The user will never know, except perhaps for an unaccustomed delay in response, where the page is coming from. How such page addresses are distributed will be clarified below. The paging module, on the other hand, is unaware of whether the page it is retrieving contains object code, data, text, digitized voice, pixels, or perhaps some specialized data structures used in the concerns of a small community spread around the world.

THE INTERACTION BETWEEN PAGES AND CONTEXTS

In discussing the objectives of natural access, we introduced the notion of an NAS context. Such contexts were introduced as encompassing vocabulary, language, definitions, database, and possibly special procedures and extensions. New contexts are created by invoking the basing operator. For example, let us say that the Context BB is based on the Context AA.

)Base BB on AA.

To understand how Contexts and pages interact, recall that the grammar table of AA is made of two parts, the binary search tree and the rule definitions, both on pages. Language extensions such as user-introduced vocabulary and applications programmer-introduced grammar rules are added to this table. As long as context BB is based on AA, new entries in the binary search part of BB's grammar table, which are copies of those in the corresponding binary search part of AA's grammar table, are introduced. The pointers in AA's binary search tree point into its rule definitions which in turn point into AA's knowledge base. On the other hand, in BB's copy of AA's binary search tree, the pointers point into AA's rule definitions. As a result, the binary search part of BB initially appears exactly like that of AA to the end user. Of course, as extensions and changes are made to BB by the user, BB's own rule definition section begins to take shape, and BB evolves into a distinct context with its own knowledge base, but always remains based on AA.

The grammar tables and knowledge base that lies under them are on pages. Thus, from the point of view of the paging module, the addressability of the knowledge base of AA from BB is insured by the fact that the page addresses of AA's rule definitions are copied into the binary search tree of BB by the basing operation.

To illustrate this point, imagine that the Sales Division of a corporation is in charge of the corporation's Eastern and Western Sales Regions. Let Central Sales be the working context of the Sales Division, containing its local records, etc. Suppose the Sales Division operator types

)Base Central Sales on Eastern Sales
)Base Central Sales on Western Sales

Now, Central Sales contains pointers to pages that reside in the offices of the Eastern Sales Region and the Western Sales Region. Their contexts are kept up-to-date by their respective staffs so that when the manager of Sales Division asks the system to

)Display a pie chart of the sales of each product line for each sales region.

the paging module in the Sales Division computer "knows" it must retrieve the necessary pages of the respective offices of the two sales regions. It also knows the file and page numbers of these pages, because these file and page numbers were passed to Central Sales at the time of basing. Roughly speaking, this is how addressability is handled for the *distributed* database of the sales organization of the given corporation. For a more complete discussion, see the work of Yu.[11]

THE SOFTWARE ENGINEERING ENVIRONMENT OF THE NATURAL ACCESS SYSTEM

To extend a user's context with a new capability, the applications programmer adds a new syntax rule and its associated semantic procedure. For example, suppose an applications programmer wishes to add a new numeric function which his user, a banker, will refer to as *projected average return*. The syntax rule to be added is

RULE proj_avg_ret
⟨num att noun⟩⇒"projected average return"
proj_avg_ret_proc

where proj_avg_ret_proc is the associated semantic procedure which the applications programmer then writes. Once this rule and procedure are added to the user's context, the user can ask the system to

)Display a bar graph of the projected average return of each of the high tech stocks of Mr. R. C. March.

Associated with each user context is a corresponding applications programmer's context which we call the *meta-context.* Figure 4 illustrates these relations in the software engineering environment.

When a context such as AA is based upon another, say BB, second contexts are created, namely meta-AA, based on meta-BB. Just as all user contexts are ultimately based upon the BASE context which contains NAS English, all meta-contexts are ultimately based on META-BASE, which contains Pascal. When the applications programmer wishes to

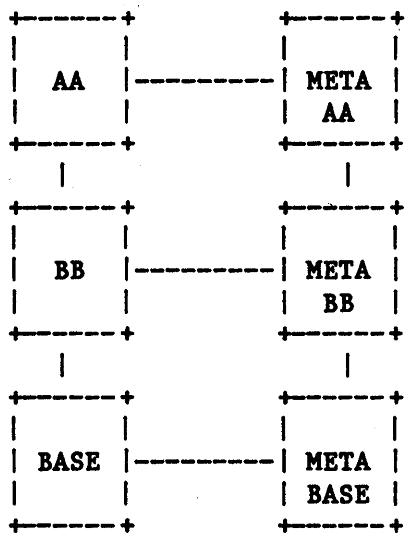


Figure 4—Contexts and their associated meta-contexts

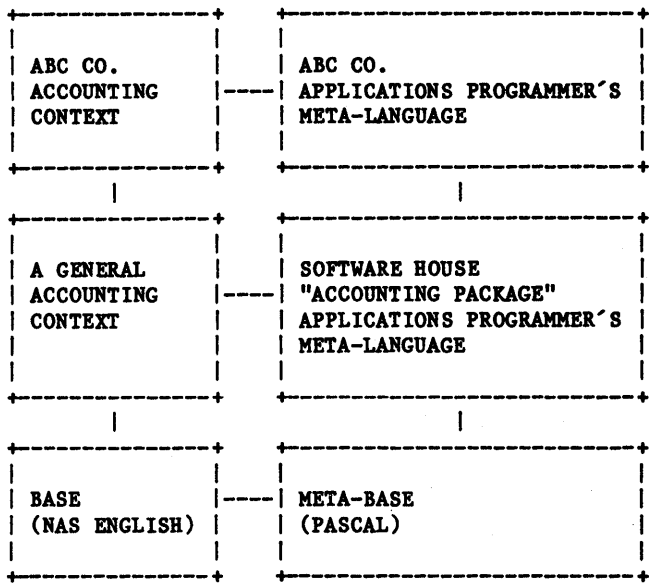

ABC CO. ACCOUNTING CONTEXT
ABC CO. APPLICATIONS PROGRAMMER'S META-LANGUAGE
A GENERAL ACCOUNTING CONTEXT
SOFTWARE HOUSE "ACCOUNTING PACKAGE" APPLICATIONS PROGRAMMER'S META-LANGUAGE
BASE (NAS ENGLISH)
META-BASE (PASCAL)

Figure 5—Steps in the applications programming chain

add a rule to Context-AA, he enters the associated meta-context, META-AA. Typing "RULE," and the name he will use in referring to this rule, he is prompted for the syntax and semantics.

```
)RULE proj_avg_ret
)Syntax: ⟨num att noun⟩⇒"projected average return"
)Semantics: procedure proj_avg_ret;
            begin
            . . .
            end.
```

The meta-context then

1. collects the semantic procedure;
2. adds the necessary INCLUDEs for system types and variables, utilities, etc.;
3. calls the Pascal compiler;
4. puts the resulting compiler-produced text as a text object in the meta-context's lexicon;
5. takes the object code and links it into the NAS resident code;
6. puts the linked code onto pages; and
7. puts the syntax into the grammar table of the user's context with the page address of the semantic procedure as "semantics."

Of course, META-BASE has a much more extensive software engineering environment.[12] It is driven by the syntax and semantic procedures of the "metalanguage," and is processed like any other NAS language by the language processor (thereby sharing such services as the spelling corrector, definitional utilities, etc.). As one might imagine, the metalanguage includes a wide range of debugging tools, and the syntax and semantics for rule addition illustrated above.

The rule-adding capability allows the applications programmer to extend his user's language. He can also extend his own language with PROCs and MACROs, thereby providing highly tailored programming environments for special application domains. In this way, application programmers who are low in the applications programming chain, as perhaps in the applications areas of computer companies, can provide wide-ranging and powerful utilities with specialized domain knowledge which can be used via basing by those on the applications programming staffs of customer companies (see Figure 5).

It is this metalanguage and its many extensions that are the interactive domain of the applications programmer. This brief introduction is intended to simply give the flavor for this environment; the experienced programmer can project many of the features that are there.

THE ROLE OF SOFTWARE DEVELOPMENT COMPANIES

Beyond the end user community and the applications programmer community, there is a third community concerned with application systems that must be taken into consideration, namely the software development companies. This community, large and rapidly growing, constitutes an important organizational mechanism for the free and competitive use of our intellectual and management resources in the information age. Under a UNIX-type system, the products of this community take the form of additional shells marketed in the form of disks and cassettes.

Corresponding products in the Natural Access environment presently consist of two main programs, one that boots the system in a new installation, and one that proceeds to run it. The main functions of booting are to format the paging data set and initialize the BASE-context grammar table with NAS English and the metalanguage. To this end, both languages are transported as syntax files, and as separate files containing object code modules for the respective resident code, non-resident utilities, and semantic procedures. System utilities at boot-time build the syntax into the grammar table, page the non-resident code, and link the two together. It is these same utilities that are invoked by the metalanguage after it has called the compiler on an applications programmer's introduction of a new rule.

It is now clear that the software development house will market its package in the form of a disk containing the syntax and object code for the semantics of the new capability it is introducing. The same utilities that are used at boot time are then called to extend a given context with these capabilities. The user, or his applications programmer, enters the meta-context associated with the context he wishes to extend, types

> ⟩Extend with ⟨file or volume name containing the application package⟩

and the new capability, completely integrated with the previously existing capabilities, vocabulary, and database of the user's context is now available. This new capability will then be passed on to new contexts that are subsequently based upon the extended context. Of course, the software company would not wish to reveal the metalanguage it has developed for producing this and possibly related packages for this application domain, nor the source code for the package's semantic procedures.

The packages that will be marketed will have a far wider range than those we see in retail computer outlets today. The Natural Access System already includes text, graphic, and image processing, all as an integrated part of NAS English, and on pages thus amenable to transmission within a NAS network. Couple this with the digitizing camera (soon expected to be available from both Canon and Sony), the processing of digitized speech, and the predicted breakthrough in both the speed and cost of digital communication,[13] and the applications possibilities proliferate beyond any bounds.

In considering the preparation of packages, one should remember that all of these processing capabilities (English, text, and image processing) can be assumed to already be in the user's context. For example, a retail department store, using the *Catalogue Preparation Metalanguage Package* it has purchased from a software company, and a digitizing camera, can quickly prepare a catalogue package for its next sale. The syntax and semantics for ordering will reflect specialized knowledge of the catalogue. The telephone number for automatic dialing is also built in. The store then distributes the catalogue to each of its charge customers in the form of a disk that includes the customer's account information and address. The customer can then type

```
⟩base catalog on BASE
⟩extend catalog from disk drive
```

and then use word processing commands to find desired items, including their pictures, and order the product(s) by typing

```
⟩ Send me a silk blouse, item number 3B64, size 14.
⟩ What color (white :::, blue ///, or beige %%%): blue
  One Anne Klein II silk sports blouse, size 14, color blue
  will be sent out today to:
  Mrs. John J. Jones
  777 Oak Street
  My Town, Maine
  $94.95 plus $5.10 tax
  will be added to your account.
  Thank you for your order.
```

CONCLUSION

The answer to the question of how to get a large natural-language system into a personal computer lies in the paging architecture of the system. The key is to use the input sentence, in conjunction with the lexicon and grammar table, to identify the minimal segments of both object code and data that must be brought into main memory. Once such a maximally-paged architecture has been effectively implemented, it has wide-ranging implications for process integration, networking, knowledge base distribution, and the software engineering environment. The Natural Access System optimizes this architecture and exploits these implications.

REFERENCES

1. Thompson, B. H., and F. B. Thompson. "Introducing ASK: A Simple Knowledgeable System." *Proceedings of the Conference on Applied Natural Language Processing, ACL and NRL.* Santa Monica, California 1983, pp. 17–24.
2. Thompson, B. H., and F.B. Thompson. "ASK Is Transportable in Half a Dozen Ways." *ACM Transactions on Office Information Systems,* April 1985.
3. Thompson, B. H., and F.B. Thompson. "Shifting to a Higher Gear in a Natural Language System." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981.
4. Thompson, B. H., and F. B. Thompson. "Customizing One's Own Interface Using English as Primary Language." *Proceedings of the South East Asia Regional Computer Conference.* Hong Kong: Hong Kong Computer Society, 1984, pp. 15.1–15.16.
5. Thompson, B. H., F. B. Thompson, and T. P. Ho. "Knowledgeable Contexts for User Interaction." *AFIPS, Proceedings of the National Computer Conference* (Vol. 52), 1983.
6. Minsky, M. "A Framework for Representing Knowledge," in P. Winston (ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill, 1975.
7. Schank, R. C., and R. P. Abelson. *Scripts, Plans, Goals and Understanding.* Hillsdale, N.J.: Lawrence Earlbaum, 1977.
8. Hendrix, G. C. "Encoding Knowledge in Partioned Networks." In N. Findler (ed.), *Associative Networks.* New York: Academic Press, 1979.
9. Trawick, D. J. "Robust Sentence Analysis and Habitability." Ph.D. dissertation, California Institute of Technology, 1983.
10. Sanouillet, R. "ASK French, a French Natural Language Syntax." Master's thesis, California Institute of Technology, 1984.
11. Yu, K. I. "Communicative Databases." Ph.D. dissertation, California Institute of Technology, 1981.
12. Thompson, B. H., and F.B. Thompson. *Natural Access to a Personal Computer,* forthcoming.
13. Sussenguth, E. H. "MIPS & BPS—Communication Abundance for All?" *Proceedings of the South East Asia Regional Computer Conference.* Hong Kong: Hong Kong Computer Society, 1984, pp. 9.1–9.16.

The need for text generation

by KATHLEEN R. McKEOWN
Columbia University
New York, New York

ABSTRACT

For a variety of systems, such as expert systems, database systems, and problem-solving systems, text generation is one way for the system to communicate effectively with its users. This is particularly true when the system is likely to be used by a wide range of users with varying levels of expertise and background. In this paper I will show why explanation is a crucial feature of expert systems, how text generation can be used within database systems to familiarize users with the database, and where text generation can aid communication with problem-solving systems. Given that text generation is more than just a frill for such systems, a second focus of the paper will be on the kinds of problems that any designer of a text generation system must address. Some of the problems include being able to decide what to say, how to organize that information, and how to express it in natural language.

INTRODUCTION

As computer systems become more sophisticated, they must be able to communicate their results successfully to their users if they are to be effective. For a variety of systems, the use of natural language text is becoming increasingly popular for such communication. This is particularly true when the system is likely to be used by a wide range of users with varying levels of expertise and background. Many potential users of complex computer systems are naive and infrequent users: They not only are unfamiliar with the computer and the formal languages available to interact with it, but their planned use of the system is infrequent enough that it does not warrant the time needed to learn a formal language. For such users, the ability to communicate with systems in everyday language promises to open the door to a world of information and tools they were previously unable to use. For expert users, on the other hand, text generation can provide a summary of complex system behavior that can allow them to detect errors before delving into system details.

In this paper I will show the role text generation plays in three types of systems: expert systems, database systems, and problem-solving systems. These systems all have a wide variety of users and have become complex enough that text generation is appropriate for the interface if ease of use is an issue. Given that text generation is necessary for such systems, the paper will also focus on the problems involved in communicating through natural language. A sketch of the kinds of solutions that have been successfully used is included, but this overview primarily serves as motivation for the text generation problem and an introduction to the solutions and issues considered by the other participants in this session.

EXPERT SYSTEMS

Communication with the user in expert systems has been needed primarily to explain the reasoning used by the system in producing its advice. Textual explanation has proved crucial to the success of expert systems for several reasons.

First, expert system users are often not computer scientists and would be unable to follow a formal representation of the system's reasoning. For example, users of medical expert systems are doctors and medical students. Natural language is a mode of communication familiar to users such as these who may not want to take the time to learn other modes.

Users, though not experts in the programming methodology of expert systems, are often experts in the domain of the system. Again, doctors fit this characterization. Their purpose in using the system is often for consultation: to gain advice on a case or to confirm their own diagnosis. To evaluate the advice provided and to determine whether to accept it, such users need to be able to understand both how and why the system came up with its advice.

Builders and maintainers of expert systems are now pointing out the value of textual explanation in identifying errors in the underlying inferencing process. Often a trace of the inference process itself can be so lengthy (for example, in ACE,[1] a single recommendation may invoke up to 15000 individual production rules) that errors are difficult to detect. Kukich[2] cites this benefit when an explanation facility was first added to the XSEL system.[3] The underlying inferencing system had been constructed incrementally by a number of different researchers who often did not understand the conventions used previously. Her explanation facility immediately pointed out even such simple discrepancies as errors due to roundoff, which had gone undetected.

DATABASE SYSTEMS

Natural-language interfaces to database systems allow a user to retrieve information from the database by asking questions and receiving answers in English. For many questions, the response can be generated by simply formatting the results of a database search in a list or as part of a sentence. It has been shown,[4] however, that many users of database systems, particularly naive and infrequent users, need to ask questions to familiarize themselves with the database before asking specific questions about its contents. Such users need to know what information is available in the database (e.g., "What kind of data do you have?"), what specific terms mean in the context of the database (e.g., "What is production cost?"), or what the differences are between different terms (e.g., "What's the difference between manufacturing and production cost?"). Questions like these typically cannot be answered by doing a search of the underlying database, and this is one place where text generation has played an important role. Natural language is particularly appropriate for answering such questions, since they require definitions, descriptions, and longer textual sequences. To generate these kinds of responses, a formulation of strategies that can be used to organize and determine content of the response is required.[5]

Research in response generation has also addressed the problem of producing responses that cooperatively address the questioner's intentions. Frequently, users reveal in their questions a presumption about the database that turns out to be incorrect. The encoding of presumptions in utterances is a formal feature of natural language that can be exploited to detect and correct a user's misconceptions.[6,7,8] If such presumptions are not corrected, the user may be left with false beliefs about the database even if all his/her questions have been answered correctly.

PROBLEM-SOLVING SYSTEMS

By problem-solving systems, I mean systems that are capable of working together with another agent (whether a human, another computer, or a robot) to solve a task. Since the two agents must work together cooperatively to solve the problem, communication between them is crucial. In this type of system it is often not necessary to generate lengthy text, but the utterances that are generated must be easily understood by the other participant. For example, in instructing the other participant which tool to use next in a task, the system must use a description that will allow the participant to pick out the correct tool.

Interactive problem-solving systems are often set up so that the system is the expert and the user is an apprentice.[9, 10] The user is not knowledgeable about how to perform the task (or solve the problem), and his/her purpose in using the system is to learn how to do so. Some of the tasks that have been considered include construction of a particular piece of equipment (such as the assembly of a water pump) that involves being able to communicate about objects and tools in the physical world. Others have been tasks that could be solved purely verbally, such as in GUS,[11] a system that acts as an airline reservation agent and helps the user select appropriate flights for a trip. Communication in natural language is appropriate for these types of systems, since the user is a novice and thus is unlikely to be familiar with a formal language developed for the domain. Furthermore, language has a long tradition of use as an interactive tool for communication. It is well suited to a situation in which participants must interact heavily to reach a solution.

PROBLEMS IN TEXT GENERATION

The previous sections have demonstrated that there is a need for communication on the part of the system in natural language in a variety of system types. The character of the generated text may differ; but natural language is an appropriate medium, in large part because of the people who will be using such systems. Given that there is a need for text generation in these different types of systems, what types of issues must a designer of a text generation system consider? How do these issues manifest themselves in the different types of systems? How close is text generation research to having practical and implemented solutions to these issues?

In the following sections, a simple but well-tried engineering approach to text generation that is currently used in many systems is first presented. Problems with this approach and issues that must be further considered in order to develop a robust and effective text generation system are then explored.

Canned Text

The simplest method for producing computer-generated text is the use of canned text and templates. This approach is probably the most commonly used method in systems requiring production of a limited amount of text. Most practical expert systems today use templates to produce explanations, and help systems are one of many examples of the use of canned text. The use of canned text requires that the system designer anticipate all questions that might be asked by the user, create the answers by hand, and store these answers as strings, which are retrieved by the system when required.

Templates are slightly more general than canned text, since they provide "text frames" that can be used to answer more than one question of the same type. Templates are English phrases with slots that can be filled in by different words for different occasions. An entire text can be produced by stringing together individual templates that each describe a step in the process. (For example, a single template is associated with each rule in an expert system, and an explanation is produced by stringing together the templates associated with the rules that fired. Slots in the templates are filled by English translations of instantiated variables in the rules.) As with canned text, templates must be produced by the designer by hand when the system is first built, and care must be taken that reasonable texts will be produced when several templates are strung together.

Canned text and templates have the advantage that their generated text can be as sophisticated as the system designer's own prose. Furthermore, it is an engineering technique that is easy to implement. There are numerous problems with this approach, however. Since the system code can be changed independently of the associated text, there is no guarantee that the generated text accurately reflects what the system actually does. An intensive personnel effort is required at the beginning of system development to hand-encode answers, and this effort must be duplicated every time a new system is developed. Finally, in large systems it may be difficult to anticipate all situations in which text will be required in advance.

Deciding What to Say

If text is not prestored ahead of time for the system to retrieve when needed, the text generation module must be able to determine what information to convey, given a request for communication. For certain questions, such as requests for definitions in the database domain, there may be a potentially large amount of information that could be used to answer the question. The system must be able to filter out information in its knowledge base that can be ignored and pinpoint information that should be included.

A number of factors can influence these decisions. The purpose for which information is required can indicate what type of information will be useful. For example, for a request for a definition, information about an object's class membership, its distinguishing attributes, examples, or analogies are appropriate. For a request about the differences between two objects, shared attributes, different class membership, and distinguishing attributes are appropriate. One technique, then, for determining what to say is to use different discourse strategies such as these for different purposes. (For more information, see McKeown.[5])

A text generation system cannot say more than it knows, as represented in its knowledge base. In order to generate particular types of text, it may be necessary to specify the type of information needed in the knowledge base. Swartout[12] shows what information must be added to an expert system knowledge to produce acceptable justifications of the system's advice. Finally, depending on who the system is talking to when a questions is asked, different information will be relevant. Appelt[10] has shown how information about the current user's beliefs should influence what the system says in order to make communication successful. Similarly, Paris[13] identifies how information about the user type (for example, whether naive or expert) can influence how much detail to include in a text.

Deciding When to Say What

Having determined what information is relevant, a text generation system must be able to order that information to produce the text. Order of a text can be crucial to a reader's understanding of it. Order alone can be used to convey temporal sequence, causality, or exemplification. Many early systems simply traced the underlying knowledge base to determine order, doing simple transformations on the underlying data structures. This method requires that the knowledge base be appropriately structured for text generation in addition to meeting all the other demands placed on it. Furthermore, while one knowledge base structure may facilitate inferencing, it may not be appropriate as a blueprint for text production.

Knowledge about discourse structure encoded as strategies can be used for determining order as well as content. Again, for situations where definitions are required, the strategy might not only dictate that class membership information and examples should be used, but also that examples should be included only after class membership has been provided. Discourse structure is currently used to help determine order of presentation in several text generation systems.[5, 13, 14]

Deciding How to Say It

The text generation system must also be able to determine what the surface text should look like. This involves making decisions about what vocabulary to use (and in particular, how to choose between synonyms), when to use a pronoun and when to use a full noun phrase to refer to an object or concept, whether to use a sequence of simple sentences or to combine several simple sentences into a single complex sentence, and how to arrange the words in each sentence. Almost all these decisions are influenced by syntactic constraints on language; thus, one component of a language generation system is a grammar. McDonald[15] describes the kinds of constraints that must be included in a grammar and how it can be used to produce fluent text. One benefit of the use of a grammar over templates is that the system can decide how to combine phrases to produce acceptable text, whereas with templates this work must be done by the system designer manually checking that templates will not produce unacceptable text when strung together.

Other influences on surface level decisions include information about the person the text is intended for and information about the discourse structure of the text. Information about user type can be used to select appropriate vocabulary (the naive user will not understand the expert's terminology). Similarly, information about the user's knowledge can be used to generate noun phrase descriptions so that the user can successfully identify what is referred to by the description.[16] Finally, knowledge about how a given sentence fits in with the rest of the text can be used to choose the best word order for a sentence and to decide whether to use pronouns.

SUMMARY

Decisions that must be made by a text generation system range over a variety of knowledge sources and are influenced by a variety of factors. Furthermore, while I have identified them as separate problems, these problems interact so that decisions often cannot be made independently.[16] Systems currently exist that have addressed each of the problems cited above. Few of these systems, however, handle more than several problems in a single implementation. The other papers in this session focus on specific problems, illustrating more sophisticated text generation techniques that are currently available beyond the limited canned text and template approach.

ACKNOWLEDGMENTS

Research in text generation at Columbia University is partially supported by ARPA grant N00039-84-C-0165 and by ONR grant N00014-82-K-0256.

REFERENCES

1. Stolfo, S., and G. Vesonder. "ACE: An Expert System Supporting Analysis and Management Decision Making." Technical Report, Department of Computer Science, Columbia University, 1982.
2. Kukich, K. "Knowledge-Based Explanation Generation." Paper presented at Second Annual Language Generation Workshop, July 8–10, 1984, Stanford University.
3. McDermott, J. "Building Expert Systems." In *Proceedings of the 1983 NYU Symposium on Artificial Intelligence Applications for Business*. New York: New York University, 1983.
4. Malhotra, A. "Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis." MAC TR-146, Massachusetts Institute of Technology, 1975.
5. McKeown, K. R. "Generating Natural Language Text in Response to Questions about Database Structure." Ph.D. dissertation, University of Pennsylvania, 1982.
6. Kaplan, S. J. "Cooperative Responses from a Portable Natural Language Database Query System." Ph.D. dissertation, University of Pennsylvania, 1979.
7. Mays, E. "Correcting Misconceptions About Data Base Structure." In *Proceedings 3rd Canadian Society for the Computational Studies of Intelligence Biennial Meeting,* Victoria, B.C., 1980.
8. McCoy, K. "Correcting Misconceptions: What To Say When the User is Mistaken." In *Proceedings of Computer and Human Interaction Conference,* 1983 Cambridge, Massachusetts, 1983.
9. Grosz, B. J. "The Representation and Use of Focus in Dialogue Understanding." Technical note 151, Stanford Research Institute, Menlo Park, California, 1977.

10. Appelt, D. E. "Planning Natural Language Utterances to Satisfy Multiple Goals." Ph.D. dissertation, Stanford University, Stanford, California, 1981.
11. Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd. "GUS, A Frame-Driven Dialog System." *Artificial Intelligence,* 8 (1977), pp. 155–173.
12. Swartout, W. "Knowledge Needed for Expert System Explanation." In *AFIPS, Proceedings of the National Computer Conference* (Vol. 54), 1985.
13. Paris, C. "Description Strategies for Naive and Expert Users." Technical Report, Department of Computer Science, Columbia University, 1984.
14. Mann, W. "Discourse Structures for Text Generation." In *Proceedings of Conference on Computational Linguistics,* 1984, Stanford University, 1984.
15. McDonald, D. D. "Surface Generation for a Variety of Applications." *AFIPS, Proceedings of the National Computer Conference* (Vol. 54), 1985.
16. Appelt, D. "Planning and Language Generation in Problem-Solving Systems." In *Proceedings of the National Computer Conference* (Vol. 54), 1985.

Knowledge needed for expert system explanation

by WILLIAM R. SWARTOUT
University of Southern California
Marina del Rey, California

ABSTRACT

Adequate explanations of the steps followed by an expert system can sometimes be produced automatically by paraphrasing the rules or methods of the system in English. However, this approach will not work for other kinds of explanations, such as justifications of the system's actions. The problem is that the basis for justifications lies in knowledge used to create the expert system, which is not represented in the system's code and hence is unavailable for explanation. This paper outlines an approach to capturing the knowledge required to provide justifications and illustrates both the problem and the solution with examples of machine-generated English text.

INTRODUCTION

Documentation of computer programs is a chronic problem. It is usually created, if at all, by a process (and an individual) not directly coupled to the development and maintenance of the program itself. Thus, the documentation is often unavailable or out of date, uninformative, and not necessarily reflective of the true content of the system. It is difficult to provide material responsive to all the many different kinds of questions that lead users and would-be maintainers to consult the documentation. Automatic text generation raises the intriguing possibility of creating systems that could provide their own documentation. In particular, it would be a significant improvement if a computer system could describe its own inner workings—both how and why it does what it does.

EARLY APPROACHES TO EXPLANATION

In the area of expert systems, some progress has been made toward achieving this goal. Some first-generation expert systems were able to provide explanations of how they performed various tasks by paraphrasing the rules or methods they used in English.[1,2] For this to be successful, the rules or methods of the system had to be written in a stylized fashion, and the techniques that the system employed had to be relatively close to those that users would be familiar with, since the system's paraphrases mirrored the code directly. If that could be done, there was a major gain: Since the explanations were provided by translation of the code itself, any changes to that code were immediately reflected in the explanations; therefore the system's documentation was always consistent with the system itself.

Unfortunately, the technique of paraphrasing the code is limited to describing what a system does or did. It cannot provide good explanations of why the system did what it did—that is, justifications of the system's actions. The problem is that the knowledge needed to support such explanations, the "rationale" behind the code, is not represented in the code itself and is hence unavailable to the paraphrasing routines.

Let us illustrate some of these problems with an example from a first-generation version of the Digitalis Therapy Advisor,[2,3] an expert system designed to help physicians in prescribing digitalis, a drug commonly given for certain cardiac problems. One aspect of digitalis therapy is that there are certain physiological abnormalities such as elevated serum calcium that may make some patients more sensitive to the drug. In Figure 1 we see the sorts of limitations that the translate-the-code approach suffers from in providing justifications. Here, the system has asked the user to enter the level of serum calcium. The user, rather than entering an answer, has asked "why?" indicating that he wants to know why the question is being asked. The advisor produced an explanation by translating its program stack. This explanation does suggest that serum calcium is a sensitizing factor for digitalis; but if the user wants to know why serum calcium makes a patient more sensitive to digitalis, he's out of luck, because that knowledge is not represented in the system. It is not represented because it does not have to be for the program to *perform* correctly. Just as one can follow a recipe and bake a cake without ever knowing why the flour or baking powder is there, so too an expert system can deliver impressive performance without any representation of the reasoning underlying its rules or methods. For the digitalis advisor to mimic expert behavior, all it needs to know is that it must ask for the value of serum calcium and reduce the digitalis dose if the serum calcium is above a certain threshold. But if we want to justify that expert behavior and explain why it is important to check serum calcium, much more is needed: The system must know that digitalis administration and elevated serum calcium have the similar effect of making the patient more likely to develop a dangerous arrhythmia and that therefore a dangerous overdose must be avoided by adjusting the digitalis dose. This gap between the knowledge needed for performance and that needed for explanation is a recurring problem whenever one tries to retrofit an explanation capability to an existing *performance* expert system. (See also Reference 4 for a related discussion.) This leads to the perhaps obvious conclusion that knowledge representation is a concern of text generation because the quality of the text produced depends in part on the underlying knowledge representation. To put it more simply: A system has to know what it's talking about. The question to be addressed is: How can the knowledge needed to justify the system be represented?

THE XPLAIN APPROACH

One way to capture the knowledge needed for justifications would be to require the system builder to annotate the system during creation with text strings justifying the system's meth-

```
WHAT IS THE LEVEL OF SERUM CALCIUM?

ENTER NUMERICAL VALUE = = = => why?
The user types "why?" indicating that he questions why he is being asked
to supply the value of serum calcium.

MY TOP GOAL IS TO BEGIN THERAPY. ONE STEP IN DOING THAT IS TO CHECK
SENSITIVITIES. I AM NOW TRYING TO CHECK SENSITIVITY DUE TO CALCIUM.
```

Figure 1—A limited justification for a question

ods. These text strings would then be displayed during execution in response to queries.[5] Clearly, the problem with this approach is that there is no way to ensure that the justifications provided accurately reflect what the code actually does. This problem is exacerbated by program maintenance and evolution. The approach we adopted in the XPLAIN system[6] depended on the observation that the *programmer* of an expert system usually had the knowledge needed to provide good justifications of its actions, because the knowledge needed for justifications was also needed to create the program in the first place. The XPLAIN system replaced the human programmer with an automatic programmer that created an expert system (in this case, portions of the Digitalis Therapy Advisor) from abstract knowledge about how the domain worked and about how problem solving was performed in that domain. As it created the expert system, its reasoning was recorded in a machine-readable form and was used to provide machine-produced justifications of the expert system's performance. The reason for using an automatic programmer is that it assured us that the resulting expert system was consistent with the underlying domain knowledge that it was based upon, and hence that the explanations the system offered actually reflected the behavior of the expert system. Although, in general, automatic programming is hard, it was more tractable for us because *algorithmically* expert systems are relatively simple.

The automatic programmer started with a single abstract goal. For example, in creating the digitalis advisor the goal was "administer digitalis." This goal was refined into more specific goals, such as "anticipate digitalis toxicity" and "assess toxicity," finally reaching the level of system primitives. (The system primitives were low-level operations that could not be further refined. They were similar to Lisp functions such as SETQ and COND.)

The automatic programmer relied on two distinct bodies of knowledge in creating the program. The *domain model* consisted of descriptive facts about the domain of the expert system, such as causal relations, associations between diseases and symptoms, and a disease hierarchy. A simplified illustration of the causal information contained in the domain model for digitalis administration is shown in Figure 2. The important thing to note is that though the domain model described important characteristics of the domain, it did not indicate how problem solving should be done: There was no knowledge about how to diagnose or treat a patient. That body of knowledge was contained in the *domain principles.* * The domain principles were used to drive the refinement process forward.

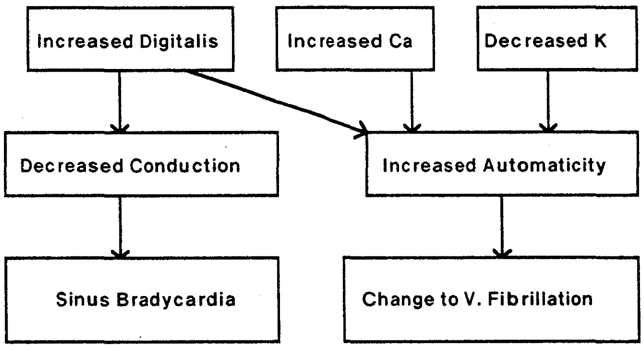


Figure 2—Simplified domain model for digitalis therapy

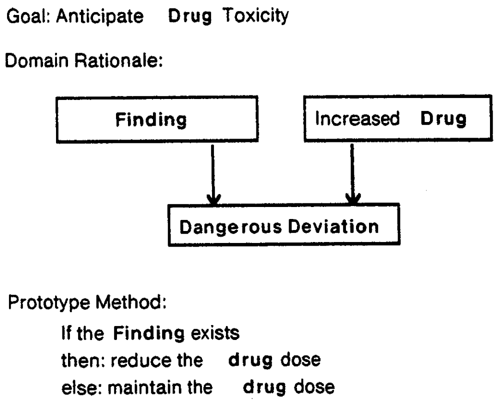


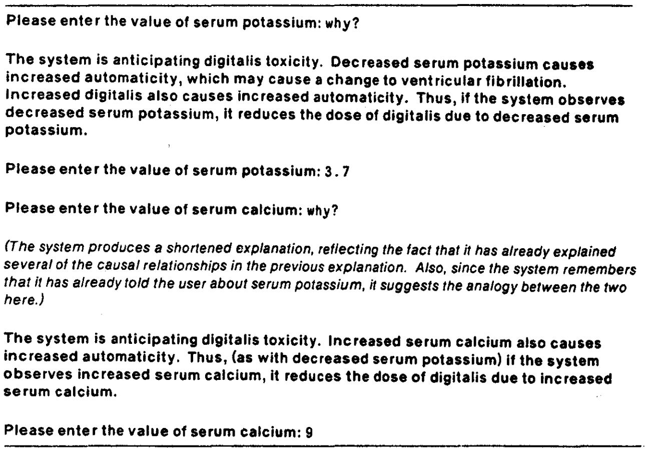
Figure 3—Principle for anticipating drug sensitivity

Domain principles had three major parts: a goal, a prototype method, and a domain rationale. The goal of a domain principle stated what its prototype method could accomplish. For example, the domain principle whose goal was "anticipate digitalis toxicity" had a prototype method that stated in lower-level terms how to do that. When the automatic programmer tried to refine a goal, it examined the domain principles to find one whose goal most closely matched the goal to be refined. In this case, the principle that matched was "anticipate drug toxicity." The match succeeded, because digitalis is a kind of drug. A simplified version of this principle appears in Figure 3. It captures the commonsense notion that if one is considering administering a drug, and there is some factor that enhances the deleterious effects of that drug, then if that factor is present in the patient, less drug should be given. Notice that this principle is not specific to digitalis therapy but could be applied to any sort of drug therapy.

Next, the domain rationale associated with the principle was matched against the domain model. In this case, the automatic programmer determined which particular findings should be checked by matching the domain rationale against the domain model. The domain rationale is the major feature that distinguished the XPLAIN approach from other refinement-based systems.[7] Its purpose is to integrate knowledge from the descriptive domain model into the refinement process by defining terms at one level of refinement with terms at the next level down.

To elaborate, refinement-based systems move through different levels of language: They refine a high-level description of a goal or problem into a low-level specific one. Yet the steps between levels of language are often quite implicit. In particular, the correspondence between terms used at one level of language and their realization at the next level down is usually implicit. The domain rationale allowed us to indicate that

* The term *domain principles* is a bit misleading. Though some of the domain principles were quite domain-specific, others (such as principles that set up backward chaining control) were largely domain-independent.



```
Please enter the value of serum potassium: why?

The system is anticipating digitalis toxicity. Decreased serum potassium causes
increased automaticity, which may cause a change to ventricular fibrillation.
Increased digitalis also causes increased automaticity. Thus, if the system observes
decreased serum potassium, it reduces the dose of digitalis due to decreased serum
potassium.

Please enter the value of serum potassium: 3.7

Please enter the value of serum calcium: why?

(The system produces a shortened explanation, reflecting the fact that it has already explained
several of the causal relationships in the previous explanation. Also, since the system remembers
that it has already told the user about serum potassium, it suggests the analogy between the two
here.)

The system is anticipating digitalis toxicity. Increased serum calcium also causes
increased automaticity. Thus, (as with decreased serum potassium) if the system
observes increased serum calcium, it reduces the dose of digitalis due to increased
serum calcium.

Please enter the value of serum calcium: 9
```

Figure 4—An explanation of why serum potassium and calcium are checked, produced by XPLAIN

correspondence. Here, for example, the high-level description of the problem is "anticipate toxicity," whereas the low-level method deals with "findings." The domain rationale defines which findings should be checked in anticipating toxicity, namely, those findings that cause something dangerous to happen that is also caused by an increased level of the drug.

In this case, there were two matches for the domain rationale: one for increased serum calcium and another for decreased serum potassium. The prototype method, which was an abstract method for accomplishing the goal, was instantiated twice, once for each match of the domain rationale. In turn, the instantiations were themselves refined; and methods were found for reducing the dose, determining whether increased serum calcium existed, and so forth. This process continued until the level of system primitives was reached.

The entire refinement process was recorded so that at runtime the system's explanation generator could produce justifications such as those in Figure 4, which shows not only what the system does, but also why serum calcium and potassium should be considered sensitivities.

The XPLAIN system has been used to implement major portions of the Digitalis Therapy Advisor, and its applicability to MYCIN has been examined.[8] It appears that MYCIN could be reimplemented by using XPLAIN and that the resulting explanations would be clearer than those produced by MYCIN, but we have not actually performed the reimplementation.

LESSONS LEARNED FROM XPLAIN

Although we were primarily concerned with providing justifications, several observations emerged during the construction of XPLAIN that relate to making it easier to modify and extend expert systems.

The separation of descriptive and problem-solving knowledge is crucial. It was necessary to separate descriptive knowledge (i.e., the domain "facts" such as causal relations and associations between symptoms and disease) and problem-solving knowledge and then record how the two were brought together to be able to provide justifications. But the separation of knowledge also has important benefits for system maintenance. In most expert system frameworks, descriptive knowledge and problem-solving knowledge are inextricably intertwined. A general principle may never be explicitly represented, but instead may be encoded as many individual rules tailored to specific situations by the particulars of the domain. (See References 4 and 8 for good discussions of this point.) The XPLAIN approach permits us to represent general principles, and we rely on the automatic programmer to use descriptive knowledge to instantiate the general principles in specific situations. This can make maintenance of expert systems substantially easier. In a conventional framework, if we want to change a general principle, we must carefully modify all the rules encoding that principle, presenting us with many opportunities for making a mistake. In XPLAIN, we just have to modify the general principle itself and rerun the automatic programmer. Additionally, since principles are represented at a more abstract level, it will be possible to export some of them to new problem areas—something that would be quite difficult in a conventional framework, due to the mixing of descriptive and problem-solving knowledge. For example, the principle for anticipating toxicity is applicable to most, if not all, drugs with side effects and could be used directly in creating an advisor for a new drug.

The creation of an expert system is more principled. In building conventional expert systems, system builders reason about the deep knowledge underlying their systems in their heads. In the XPLAIN approach, the system builder is encouraged to represent deep knowledge explicitly so that the automatic programmer can reason about it mechanically. This method makes the system builder think more closely about the domain knowledge and problem-solving strategies adopted. The result is that flaws are found that might otherwise be overlooked. For example, an early version of the digitalis advisor, written before our work on XPLAIN, treated abnormal thyroid function (myxedema) as a sensitizing factor for digitalis. When we reimplemented the advisor, using XPLAIN, we found that myxedema did not seem to meet the requirements imposed by the domain principles for something to be considered a sensitivity. A careful examination of the medical literature revealed that in fact, myxedema is not a sensitivity; instead, it impedes digitalis excretion. (See Reference 6, Section 7.1.) Thus, by imposing some additional structure on the way an expert system is constructed, the XPLAIN approach can result in a sounder, more principled system.

CURRENT WORK

The Explainable Expert Systems (EES) project at ISI[9] is extending the XPLAIN paradigm. In particular, we are trying to represent additional kinds of knowledge needed to support additional kinds of questions. A broad view of the EES system is shown in Figure 5.

We provide system builders with a more expressive knowledge base that encourages them to represent knowledge more abstractly and explicitly separate out the different kinds of knowledge that go into an expert system. Like XPLAIN, EES will make use of a *domain model* and a set of *domain principles*. In addition, we will model additional kinds of knowl-

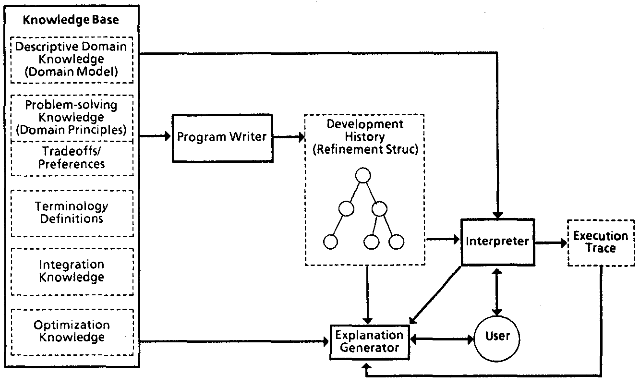


Figure 5—Global view of the EES framework

edge that will allow the system to answer additional kinds of questions. *Tradeoffs* are associated with domain principles to indicate the beneficial and harmful effects of selecting a particular strategy to achieve a goal. *Preferences* are associated with goals and are used to set priorities based on tradeoffs. Together, tradeoffs and preferences will provide the system with the knowledge it needs to explain the choices it made among alternative problem-solving strategies. Mappings between abstract terms and the concepts that realize them (which had been included as part of domain principles in XPLAIN) are broken out as a separate type of knowledge to allow *terminology* to be shared across domain principles. *Integration knowledge* is used to resolve potential conflicts among knowledge sources. *Optimization knowledge* represents ways of efficiently controlling the execution of the derived expert system and must be explicit to explain possibly unusual orderings of actions that may result from efficiency considerations.

Similar to XPLAIN, the EES framework will generate a runnable expert system by applying the *program writer* component to the knowledge base. The steps taken in producing code are recorded in a *development history,* a lattice structure whose leaf nodes represent system implementation code and whose interior nodes represent goals and decisions made on the way to generating the implementation. The *interpreter* executes the leaf nodes of the development history. It produces an *execution trace* and manages the system's normal interaction with users. User questions, however, are sent to the *explanation generator,* which accesses the knowledge base, development history, interpreter, and execution trace in the course of constructing answers to queries.

SUMMARY

We have argued that one of the failings of current explanations of expert systems is that they cannot justify what the system does because the knowledge needed to support such explanations is missing. The XPLAIN system captured that reasoning by using an automatic programmer to create an expert system and recording the decisions it went through while creating the system so that they would be available later to provide justifications. We observed that the organization this approach imposes can also be beneficial to the maintenance, evolvability, and consistency of an expert system. Our current work is extending the XPLAIN paradigm to represent additional kinds of knowledge needed to support additional explanations.

ACKNOWLEDGMENTS

The research described here was supported under DARPA Grant #MDA 903-81-C-0335 and National Institutes of Health Grant #1 P01 LM 03374-01 from the National Library of Medicine.

I would like to thank Bob Balzer, Jeff Barnett, Jack Mostow, Bob Neches, Ramesh Patil, and Peter Szolovits for many enlightening discussions that led to the ideas discussed here.

REFERENCES

1. Buchanan, Bruce G., and Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, Mass.: Addison-Wesley, 1984.
2. Swartout, W. R. *A Digitalis Therapy Advisor with Explanations.* Technical Report TR-176, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1977.
3. Gorry, G. A., H. Silverman, and S. G. Pauker. "Capturing Clinical Expertise: A Computer Program that Considers Clinical Responses to Digitalis." *American Journal of Medicine,* 64 (1978), pp. 452–460.
4. Clancey, W. "The Epistemology of a Rule-Based Expert System—A Framework for Explanation." *Artificial Intelligence,* 20 (1983), pp. 215–251.
5. Chandrasekaran, B., and S. Mittal. "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving." In *AAAI Proceedings of the National Conference on Artificial Intelligence.* William Kaufman, Inc., Menlo Park, CA, 1982.
6. Swartout, W. "XPLAIN: A System for Creating and Explaining Expert Consulting Systems." *Artificial Intelligence,* 21 (1983), pp. 285–325. (Also available as Information Sciences Institute publication RS-83-4.)
7. Sacerdoti, E. *A Structure for Plans and Behavior.* Technical Report TN-109, Stanford Research Institute, Menlo Park, CA, 1975.
8. Szolovits, P. "Toward More Perspicuous Expert System Organization." In "Report on Workshop on Automated Explanation Production," W. Swartout (ed.), *SIGART Newsletter,* ACM, July, 1983.
9. Neches, R., W. Swartout, and J. Moore. "Enhanced Maintenance and Explanation of Expert Systems through Explicit Models of Their Development." In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems.* IEEE, Silver Spring, MD, December 1984.

Planning and language generation in problem-solving systems

by DOUGLAS E. APPELT
SRI International
Menlo Park, California

ABSTRACT

Language generation bears a multifaceted relationship to problem-solving systems. If a problem-solving system is to be useful, it must interact with a user to first define the problem that is to be solved, then to convince him that the solution found is correct. Both activities require that the system's interaction with the user be planned so that the objectives of problem definition and explanation are met. Therefore, communication can be viewed essentially as a problem-solving activity that takes place within a problem-solving system. Reasoning about goals and the actions that satisfy them is an important aspect of utterance planning at the subsentential as well as the supersentential level. Language generation itself can thus be regarded as a problem-solving activity. The demands imposed by communication and language generation imply that the language generation system is not merely a back end, but an integrated part of the problem-solving system.

THE ROLE OF LANGUAGE GENERATION IN PROBLEM-SOLVING SYSTEMS

A problem-solving system essentially is one that accepts some sort of problem and, after performing some presumably complex reasoning, provides a solution. In artificial intelligence, it is usually assumed that problem solving requires symbolic inference in some sort of representation system. Although a system that solves simultaneous equations may "solve problems," it is not a really a "problem-solving system" in the sense in which artificial intelligence researchers use the term. Many problem-solving and expert systems have been developed; all have focused primarily on problem solving in the domains for which they were designed. They have barely begun to exploit natural language for interaction with the user.

The use of natural language to interact with the user is a more interesting and difficult task in problem-solving systems than it is in interactive systems, such as database query systems, which are not oriented toward problem solving. This is because problem solving requires specification of the problem to be solved and explanation and justification of the solution.

Analysis of conversations between advice seekers and expert human problem solvers reveals that the process of defining the problem is a significant problem-solving task in itself.[1] Questioners do not always know what information they need to acquire in order to achieve their goals, and sometimes fail to ask the appropriate questions. To be helpful to users, a problem-solving system should identify inappropriate questions and the expected goals. This allows the systems to supply the user with the information he actually needs.

By the same token, a simple true–false response from the system is unlikely to satisfy a user. If the problem-solving task is complex, and if the result has significant consequences, the user is unlikely to accept the response without having adequate assurance that it is correct. This implies that the system must be capable of explaining and justifying its responses. Providing an adequate explanation can require reasoning about the user's relevant domain knowledge, determining what inference steps in solving the problem need to be explained, and organizing the body of relevant information into a coherent explanation. As is the case with defining goals, explaining the problem itself can be a complex problem-solving activity.

THE ROLE OF PROBLEM SOLVING IN LANGUAGE GENERATION

One product of our intuition about language is the idea that it is an information conduit wherein sentences can be viewed as packages containing ideas.[2] Language generation, therefore, consists of wrapping an idea in an appropriate package for delivery and understanding consists of unwrapping the package to reveal the enclosed idea.

In contrast with this "conduit" metaphor, speech act theorists propose that rather than being packages containing ideas, utterances are properly viewed as actions that are performed with the intention of achieving specific objectives, much the same as hammering a nail or pouring a glass of milk. Instead of changing the physical state of the world, speech acts alter the knowledge and intentions of speakers and hearers. Planning communication should therefore have much in common with planning other kinds of actions, and the theory of problem solving in artificial intelligence should be applicable to communicative actions as well. This observation lies at the heart of utterance-planning research.[3,4]

Research in utterance planning seeks to explain the connection between the sentences that are uttered by a speaker and their intended effects on the hearer. This research has revealed that the connection is not a straightforward pairing between speech acts and the utterances that are planned to realize them. It is possible for a single utterance to achieve several goals simultaneously and thus to carry out several speech acts at the same time. On the other hand, it is also possible to analyze one speech act as being performed in the course of several utterances.

As an example of multiple communicative acts in a single utterance, consider a situation in which an expert and an apprentice are cooperating on the task of disassembling an air compressor. The expert says to the apprentice, "Use the wheelpuller to remove the flywheel," as he points to the wheelpuller. The expert is first, making a request to remove the flywheel; second, informing the apprentice that a particular tool should be used for the task; and finally, informing the apprentice that the tool is called a wheelpuller.

As an example of a multi-utterance request, consider the following dialogue:[5]

E: There is a little yellow piece of rubber...
A: Little yellow piece of rubber...
E: Next to the pump handle.
A: Yes.
E: Place it in the valve mount.

In this conversation, speaker E is, in addition to identifying an object, making a request that A perform an action. Unlike the earlier example in which, by means of a gesture, the speaker implicitly communicates the information required by the hearer to identify the necessary object, the speaker here isolates the identification request explicitly in the first utterance. The entire request is not consummated until the final utterance.

The behavior exhibited by the speaker in each of the above two examples is appropriate to different situations. In the first situation, the utterance that is produced is very efficient—no effort is wasted. This kind of behavior is most evident when communication is difficult relative to the domain of the problem-solving task. Such situations may arise, for example, when a speaker and hearer are communicating over a low-bandwidth link, such as a teletype line.

The second dialogue, on the other hand, is most appropriately conducted face to face. In this case the speaker anticipates that the task of identifying the referent will require relatively more effort, perhaps even an entire subdialogue, and that it therefore warrants placing the problem of referent identification on a level equal to that of the domain problem. To communicate effectively, a language generation system should have the ability to exhibit the behavior of the speaker in both of the two previous examples.

THE IMPLICATIONS OF UTTERANCE-PLANNING THEORY FOR THE DESIGN OF PROBLEM-SOLVING SYSTEMS

Language generation is necessarily focused on producing sentences because the sentence is the fundamental unit of linguistic analysis. The context of an utterance can determine its appropriateness, but the constraints that determine an utterance's grammatical correctness are far more localized, involving only a single sentence or, in the case of ellipsis, perhaps the preceding sentence. Therefore, it is very tempting to design the architecture of a language generation system to employ sentence production as a demarcation between two distinct processes. One process, often labeled *strategic,* is devoted to planning communication, whereas the other process, commonly called *tactical,* is devoted to grammatical realization of the output of the strategic component.

This analysis is attractive for a number of reasons. First, it provides at least two levels of abstraction; details that are relevant to the tactical level can be ignored at the strategic level. Clearly, deciding which determiner to use in an embedded clause is not an appropriate consideration at the time one is devising a strategy to persuade the hearer to believe some proposition. Second, if the interface between the tactical and strategic components were carefully specified, it would appear possible to design a general tactical component that could be used for a variety of applications.

However, the two examples cited offer some evidence that the division of processing into distinct strategic and tactical steps at the sentential level will inhibit certain types of communication planning. The examples show that communication planning, especially with respect to referring expressions, cuts across the boundary between the two conceptual areas traditionally regarded as strategic and tactical. The decision to refer to a particular individual is usually regarded as strategic; deciding what descriptors to use in the referring expression is viewed as tactical. The first example illustrates that physical actions, usually considered at a very high level and outside the scope of the grammar, can be planned as the result of tactical criteria. The speaker may have realized that the only way to identify the wheelpuller orally is with a long and complicated description based on its physical attributes. He therefore chooses nonverbal means to communicate his intention to refer. The decision to inform the hearer that the object being referred to is called a wheelpuller may be motivated by the tactical consideration of facilitating future references to the object.

In the second example, considerations of the way to refer to an object, normally a tactical problem, enter at the strategic level of determining the content of the sentences to be produced. The task of identification is separated from the main utterance of the request because the speaker is not sure the hearer will be able to identify the description without help or considerable effort on the hearer's part.

Consideration of the complex relationships between strategic and tactical processing in language generation and communication planning has led to development of the KAMP planner and the TELEGRAM grammar.[6] KAMP maintains the useful abstraction hierarchy of the strategic–tactical distinction, but eliminates any processing distinction. Grammatical knowledge is separated from world knowledge and is expressed in its own suitable notation. However, the process that employs the grammatical knowledge is not isolated from the planning process by a narrow channel of communication. Acting as coroutines, strategic and tactical processes share control, which is transferred back and forth between them; each is able to access and influence the other when it has control.

The use of this control strategy implies that the communication planner must be designed with due attention to the interface between the planner and the grammar. Furthermore, the actions involved in communication planning are determined to a great extent by what is possible in the problem-solving domain. For example, in some domains, problem solvers may be able to use physical actions; in others, they may not. Problem solvers may or may not have available other avenues of communication, such as graphics. When planning to define or explain a problem, it often is necessary to be able to recognize the user's intentions in the problem-solving domain. This may require that the full expertise of the domain problem solver be available to the communication planner. This in turn implies that the problem-solving system should be designed to render its knowledge maximally accessible.

The requirements of communication and language generation place a considerable burden on the design of the problem-solving system because the communication planner must be thoroughly integrated with both the grammar and the knowledge from the problem-solving domain. The designers of such systems would do well to bear these requirements in mind if the systems are to communicate effectively with their users.

ACKNOWLEDGMENT

This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-K-0078 with the Naval Electronic Systems Command. It was also made possible in part by a gift from the System Development Foundation. The views and conclusions expressed in this article are strictly those of the author.

REFERENCES

1. Pollack, M., and J. Hirschberg. "User Participation in the Reasoning Processes of Expert Systems." *Proceedings of the National Conference on Artificial Intelligence*. William Kaufman, Inc., Los Altos, Calif., 1982, pp. 358–361.
2. Reddy, M. J. "The Conduit Metaphor—A Case of Frame Conflict in Our Language about Language." In Ortony (ed.), *Metaphor and Thought*. Cambridge, U.K.: Cambridge University Press, 1979.
3. Appelt, D. E. *Planning English Sentences*. Cambridge, U.K.: Cambridge University Press, 1985.
4. Cohen, P., and C. R. Perrault. "Elements of a Plan-Based Theory of Speech Acts." *Cognitive Science,* 3 (1979), pp. 177–212.
5. Cohen, P. "Referring as Requesting." *Proceedings of the Tenth International Conference on Computational Linguistics*. 1984, pp. 207–211.
6. Appelt, D. E. "TELEGRAM: A Grammar Formalism for Language Planning." *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. William Kaufman, Inc., Los Altos, Calif., 1983, pp. 595–599.

Surface generation for a variety of applications

by DAVID D. McDONALD
University of Massachusetts
Amherst, Massachusetts

ABSTRACT

If a knowledge-based computer program is to produce English texts that are fluent and appropriate to the context of their audience and situation, the program's model of the conceptual information to be communicated must be augmented by a model of the English language, including linguistic information of the kind one finds in grammars and books of rhetoric, such as its constraints, alternative formulations, styles, carrying capacity, etc. Since this information is for the most part uniform across subject matters and tasks, it is included as part of any task-specific generation system, thereby relieving other designers of the need to be concerned with its unavoidable complexities. This paper describes the design issues that arise in developing such a linguistic component, using illustrations from a system that has been successfully interfaced to eight text-planning programs, and a variety of knowledge representation formalisms.

INTRODUCTION

When we want to have a conventional program, such as a compiler or an operating system, give a message to its human user, our job is straightforward; we decide where the message should go in the program's code and insert the text that we want to use, perhaps including some formatting statements or some fill-in-the-blank variables that will be substituted for at runtime with a simple name, number, label name, etc. We can get away with this simple a procedure because we can anticipate the exact situations in which our conventional program is going to have to give its messages, and so can decide once and for all on how its texts should read. We can be as polished as we like, and have no difficulty ensuring that the texts contain the right information or are grammatical because we are writing them ourselves.

Messages from knowledge-based systems, especially those that are intended to act as assistants or expert advisors, cannot be handled so easily. The simple regimen of pre-selecting what to say and including it directly in the program breaks down in the face of the impossibly large number of subtly varying situations that such a system will have to work in.

A well designed AI system will be sensitive to differences in the goals and experience of the people using it, to variations in the relative importance of the parts of what it has to say and the perspective from which they should be described, and to the content and rhetorical structure of the discourse that has come before. These factors are crucial components of what are coming to be called *intelligent interfaces,* and are beginning to be part of the criteria by which we evaluate any knowledge-based system. If its communications are not clear, fluent, and appropriate to the situations and audience, the system will not be accepted by its users.

Achieving good communications involves many factors, such as an expressive underlying representation, a good model of the user, a well-designed graphic interface, and careful planning including deliberations over alternative presentations and their consequences. This paper considers an additional factor, that of understanding linguistic facts and the process of producing utterances in a natural language.

Linguistic facts are elaborate, complexly interrelated, and not at all related to the nature of the information being communicated. Thus, the task of designing an efficient linguistics component, the part of the system that is responsible for the realization of the system's goals and internally represented information as a cohesive grammatical text, is exceptionally difficult. To make progress in a way that can be generalized, one generally has to bring together the specialized knowledge of linguists, AI specialists in representation, and sophisticated symbolic programmers. One should also not be in a hurry. Only a few research groups have ever developed programs that could generate high quality texts by general methods,[1] and the task is generally acknowledged to be more difficult than its obverse, natural language comprehension.

The difficulty the abovementioned problems pose is mitigated by the fact that if one is careful, the task need only be done once. Linguistic facts and the heuristics for using them are uniform across subject matters and tasks, thereby making it possible to capture this information in a common program module that can then be included as a part of any task-specific communications system, thus relieving other designers of the need to be concerned with its complexities. I have developed such a "linguistics component,"[2,3] and with my colleagues at the University of Massachusetts, continue to extend its abilities and apply it to tasks such as scene description,[4] tutoring,[5] and summarizing narratives.[6] In this paper we will look at the place of a linguistics component (LC) within a knowledge-based system with an intelligent interface, consider the kinds of tasks that it performs, look at diagrams of some of the information it uses, and close by examining the character of the interface between an LC and the rest of the system.

WHAT A LINGUISTICS COMPONENT KNOWS

The Position of the Component within the Larger System

Figure 1 shows the model that we use. The LC is a part of the user/system interface and is positioned analogously to that of the natural language comprehension module. Between the LC and the system proper, but still within the interface, is a text planning module. The LC and this planner are the pipeline through which information flows from the system to its user; they act as transducers, interpreting, transforming, and passing on information and intentions that originate within the decision-making apparatus of the knowledge-based system (KBS).

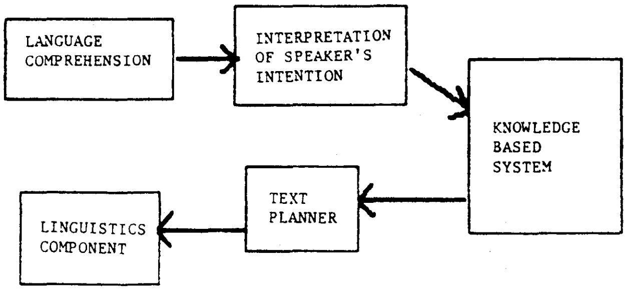


Figure 1—The place of the linguistics component

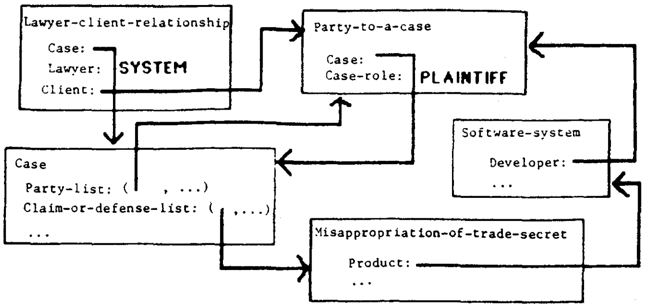


Figure 2—An example specification for a text

The KBS produces a specification, couched in the KBS's representation language, of the information to be given, and the "communicative force" it is to have. Figure 2 is an example of one of these specifications (note that most frames have more slots than are displayed in this figure). In this example, there is no special force to this specification beyond "communicate this information." The KBS plays the role of an attorney beginning to describe a case.

The effect is the same whether the specification is a new structure produced by the KBS specifically for the occasion, or whether it consists of relations among already existing objects in the KBS's knowledge base (as in this case). The specification is examined (and possibly modified and annotated) by the text planner, then passed incrementally to the LC for realization as a proper English text.

Text planning plays a crucial mediating role in its position between the KBS and the LC (see the work of McDonald,[4] Woolf and McDonald,[5] and Cook, Lehnert and McDonald[6] for detailed discussion). Suffice it to say, the principal responsibilities of this intermediate component are to determine what parts of the specified information can and should be omitted from the text because the user will infer them from other things that are said, and to organize the information in the best manner for presentation in textual form. This entails adding rhetorical specifications, determining the sequence in which its information will be presented, and making some of the decisions about what words will be used.

What Tasks the Linguistics Component Takes On

In the task context that the specification of Figure 2 normally has in our research project, the text generated for it is as follows.

> I represent a client named RCAVictim who wants to sue SWIPE Inc. and Leroy Soleil for misappropriating trade secrets in connection with software developed by my client.

This is the *unmarked* realization produced without any additional annotation of the specification by the text planner; its form thus derives entirely from the actions of the LC, given its present default settings. What have these actions done? A look at Figure 2 shows that nothing very profound has been done in the way of selecting English words for terms in the representation. Nearly every word in the output text has a counterpart in the name of a slot or a frame. This partly reflects our concentration on structural and grammatical problems rather than on concept-to-word mapping; it also reflects the genuine preference that we have for working with representation languages that encourage having a large number of very specific primitive terms whose interrelations and commonalities are given by explicit (and possibly redundant) relations, rather than with languages that encourage the decomposition of concepts into complex expressions involving a very small number of primitives.

Choice of vocabulary aside, the contributions of the LC to our text involve the following categories of "linguistic" decisions and background knowledge.

Style

The five framed KBS specification was realized as a single sentence; this, and the decision to say *a client named* and *in connection with* is a matter of adhering to a particular prose style, in this case that of one lawyer in consultation with another. By definition, choices of wording or sentence construction that are dictated by style cannot be determined by conceptual meaning (i.e., they will not follow from anything in the specification from the KBS). The style is instead determined by a concept-independent set of rules maintained in the LC and triggered by purely linguistic descriptions of the growing text (e.g., *prefer complex over simple sentences,* or *prefer role-related descriptions over actual names*).

Syntactic and morphological details

The conceptual-level representation maintained in the KBS is obviously not organized in the way that a natural language text is; once mapped into English words (mostly by the prior planning component), it must be linearized, morphologically marked, and if speech is to be produced, it must include its syntactic organization into clauses and phrases in order to initiate the correct intonational patterns. Nongrammatically-based generation components, for example those using the *direct production* technique typical in present KBSs,[1] can accomplish linearization through the use of templates. However, morphological markings are quite difficult to achieve by ad-hoc means; even making subject and verb agree can be a programming ordeal, and marking phrases is impossible.

In a linguistically knowledgeable system like our LC, a thorough syntactic description of the text is constructed as an integral part of its generation. Figure 3 shows a portion of the syntactic tree that is constructed for the example text. The bulk of its detail is intended to constrain and direct the decision-making process that selects texts for incremental parts of the KBS specification. It serves a second purpose of indicating the morphological markings that the words should have. The label *marked-gerundive-complement* will force the verb it dominates (*misappropriate*) to take on its *-ing* form when it is spoken; in other contexts, alternative labels would force the infinitive, present tense, etc., all without any other part of the LC needing to be aware of the process. So-called grammatical *function words* such as *to* or *for* are introduced

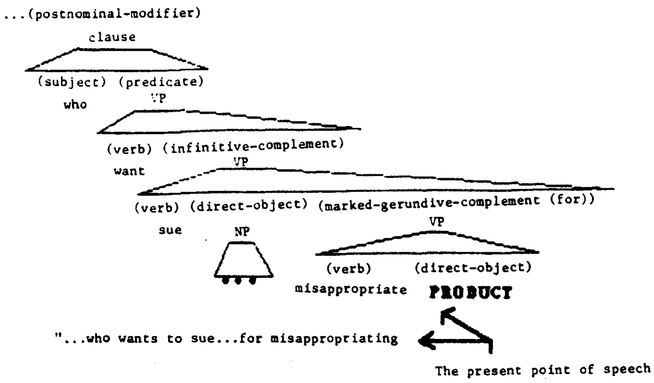

...(postnominal-modifier)
clause
(subject) (predicate)
who
VP
(verb) (infinitive-complement)
want
VP
(verb) (direct-object) (marked-gerundive-complement (for))
sue
NP
VP
(verb) (direct-object)
misappropriate
PRODUCT
"...who wants to sue...for misappropriating
The present point of speech

Figure 3—A snapshot of the LC's representation

into the text stream as a reflex of its linguistic description without having to be deliberated over anywhere else.

Grammatically-forced restructuring of information

When concepts are mapped into words, the particular choice of wording that is made, and the crucial syntactic context in which it appears will constrain what information must be included or left out. A text planning system sometimes needs to be aware of these constraints when it chooses the words (for which purpose they should be simply stated in an easily manipulated representation), but more often the differences will not matter and should be left to the LC to carry out automatically. For example, when a verb appears in a main clause, it must carry information about the placement of its event in time that cannot be included when it is embedded as a complement, regardless of whether it was important to the KBS specification or not. The subject of a verb must be omitted in many complements, or sometimes a later argument suppressed because it is linked to a grammatically strategic element in a question or conjunction. From the KBS point of view, such things are details that it should not have to worry about. The rich linguistic representation that is used within the LC makes this possible by associating editing procedures, operating in the background, with the relevant terms in the representation and having them perform the restructuring "sight unseen" by the other components of the system.

Manifesting the speaker's intent

Different text structures invariably carry with them different implications as to the relative importance or salience within the overall message of the information they convey. We know from our own writing that the same information should be expressed in different ways depending on the point we are trying to make. The question then is, how much knowledge of syntactic idiosyncracies does the text planner or the KBS need to have in order to ensure that the most appropriate text structure will be used? With a well-designed LC, the answer to this question is "none at all," because the LC will be using just this kind of knowledge to interpret the intentions given in the KBS specification, and to direct the selection of the text structure.

With this kind of division of labor, the KBS is able to couch its specification using terms like *emphasize, contrast,* or *indicate similarity of relationship*. The LC will understand these directives in terms of their implications for how the text is to be organized, for example, shifting the grammatical role of an object or choosing a relative clause over a new sentence as needed.

Maintaining cohesion in a discourse

Once a program needs to produce texts that are longer than a single sentence, or once the conversation consists of more than simply answering isolated questions and earlier statements cannot simply be forgotten, then a new kind of "grammaticality" becomes important, namely the manifestation through choice of wording and text structure of the links between the content of the present utterance and those that have preceded it. Having initially referred to *a client named RCAVICTIM,* for example, the next reference to that KBS object should use the same kind of description (e.g., *my client*) or else it will either confuse the audience as to who is being referred to, or draw unintended attention to itself.

A well-designed LC can do this completely on its own without requiring the KBS to maintain any awareness of the discourse context, or to change the way it organizes its specifications. The LC maintains a representation of the relevant parts of the earlier discourse that is analogous to its representation of the text structure it is working on (Figure 3). It uses the former representation to decide when pronouns should be used or when descriptions should be abbreviated because part of their information will still be remembered. These rules of coherency are folded into the rules for selecting text structures by the writer of the LC, and participate in the text-level decision-making on an equal footing with the intentional directives.

Why the Lack of a Linguistics Component Hasn't Bothered Most KBSs to Date

Only a few KBSs presently incorporate an LC, and probably only because the researchers who built them have had an independent interest in the problems of language generation. Present systems rely instead on template-based direct production from their internal representations along with as much ad-hoc grammar as their programmers have time to accomplish. This has worked thus far only because it has had to work (there have not been off-the-shelf LCs available for incorporation with minimal effort), and because the demands that actual KBS developers have placed on natural language output have been minimal; present KBSs still carry out only one or two system/user tasks, thus, their internally represented objects and relations have only needed to be presented in a single way which could then be captured in a template. Because the concepts that present KBSs have employed call for only simple syntactic realizations (e.g., there have been almost no systems yet that have had to talk about *expecting* or *persuading*), the syntactic contexts that their text templates impose on the variables that get substituted into them have been very simple and have not called for any sophisticated syntactical transformations from their normal format.

Without an LC you can instantiate templates, but not easily blend them together because the complexity of the grammatical adjustments that have to be made in order to keep the template combination grammatical goes well beyond what can be achieved in an ad-hoc treatment. This problem, which applies to some of the substitutions that would be wanted in instantiating templates, too, is probably the most serious limitation on the template style of language generation. These two reasons will be valid only a little while longer, however, so the designers of KBSs should begin to consider what it would be like to work with a full-scale LC.

INTERFACING TO A LINGUISTICS COMPONENT

If the builders of KBSs are to gain from the eventual availability of some off-the-shelf linguistics component from another research group, they will need to understand the kind of constraints that the LC is going to place on their designs. Assuming that the (sometimes overriding) mechanical interfacing constraints have been met (e.g., both KBS and LC are available in the same programming language dialect and consequently can be loaded simultaneously into the same program image), the remaining constraints are of two kinds, formal and ontological.

By *formal constraints,* I mean restrictions or preferences on the kinds of representational structures that are used for the KBS specification of what it wants to be communicated. We can always assume that the purely data-structure level compatibility of the interface will be trivially taken care of (e.g., we will not care whether a specification is given as a list of symbols, an array, a structure of flavor instances, etc.). For operations at that level, the LC will have defined a set of *interface functions* to do the actual movement of data between components; each new KBS will supply its own definitions for those functions, thus abstracting away from the idiosyncratic implementation decisions that went into programming the components.

The more substantial representational questions remaining can be thought about in terms of the implications of the choice of *representation language* within the KBS; different established languages fall into different categories according to whether their properties make them easy or difficult for an LC to deal with. In the case of my LC, experience with a wide range of representation languages including the predicate calculus, Lisp symbols and property lists, PLANNER-style assertions, the "frame" languages OWL and FRL, KL-ONE, home-brew type/token systems based on Lisp structures, a part/whole hierarchy built out of Flavors, and a version of RLL has been acquired.[4] Two conclusions can be drawn from this body of experience; first, that object-based representations are preferable to expression-based representations; and second, that an efficiently implemented specialization hierarchy and meta-level are great time-savers.

The difficulty with expression-based representations is that no single manipulable locus can be identified with each of the represented entities or relations in the system; instead, their identity is distributed across a set of otherwise disconnected expressions. In order to manipulate such *distributed objects,* an LC must continually parse and reparse them, or else maintain a redundant shadow representation for them on its own. An LC needs to continually ask itself questions of the general form, "Have I seen this entity before and if so in what linguistic context?" Such questions are quite burdensome when working with a representation that does not maintain a unique first class object for each item in its world.

A facility for representing meta-level relationships allows an LC to record generation-specific information that is applicable to patterns in the way entities are represented by reifying the pattern and making it an object like those at the ground level. A *specialization* or *AKO hierarchy* provides a similar service by allowing generation tactics that apply to a great many related entities to be stated compactly, with variations captured through parameterization and specialization of associated generation information in parallel with those of the entities in the domain.

Weaknesses in the formal structure of a representation (from a generation perspective) will make dealing with the representation awkward for an LC, but not impossible; however, weaknesses in the ontology that it supplies for the world it is representing can impose severe restrictions on the quality and fluency of the language an LC can produce for it. Natural language has evolved to carry the communicative load that humans place on it, and to make it easy to express the relationships and attributes that humans see in the world and consider important. If a KBS does not have the same breadth of knowledge that a person does for a given subject matter, it will be at a loss when the language demands a value for an ontological category that people always know, but the KBS does not; you cannot talk about things you don't know. All KBSs, regardless of their choice of representation language, will stumble over this point indefinitely because humans support a much richer system of distinctions and kinds in their thought and use of language than any yet captured in a knowledge base. The solution to this problem will not come from technical improvements in the design of LCs, but is an issue to be resolved by philosophers, psychologists, and people doing basic AI work on knowledge representation.

REFERENCES

1. Mann, W., G. Bates, B. Grosz, D. McDonald, K. McKeown, and W. Swartout. "The Prospects for Research in Natural Language Generation." *American Journal of Computational Linguistics,* 8-2 (1982), pp. 62–69.
2. McDonald, D. "Language Generation: The Linguistics Component." *5th International Joint Conference on Artificial Intelligence.* Available from William Kaufmann, Inc., Los Altos, California. 1977, p. 142.
3. McDonald, D., and J. Conklin. "The Use of Salience in the Generation of Natural Language Descriptions of Visual Scenes." *National Conference on Artificial Intelligence.* Available from William Kaufmann, Inc., Los Altos, California. 1982, pp. 172–174.
4. McDonald, D. "Natural Language Generation as a Computational Problem: an Introduction," in M. Brady and R. Berwick (eds.) *Computational Theories of Discourse.* Cambridge: MIT Press, 1983.
5. Woolf, B., and D. McDonald. "Building a Computer Tutor: Design Issues." *IEEE Computer,* 17-9 (1984), pp. 61–73.
6. Cook, M., W. Lehnert, and D. McDonald. "Conveying Implicit Context in Narrative Summaries." *Proceedings of the 10th International Conference on Computational Linguistics.* Association for Computational Linguistics, % Bell Communications Research, Morristown, New Jersey. 1984, pp. 5–7.

MEDCAT: An interactive computer program for medical diagnosis, consultation, and teaching

by W. D. HAGAMEN, MARTIN GARDY, GREGORY BELL, EDWIN REKOSH, and STEVEN ZATZ
Cornell University Medical College
New York, New York

ABSTRACT

MEDCAT is a computer program that makes diagnoses, explains each step in its reasoning in response to questions, increases its knowledge and reasoning ability by conversing with expert physicians, and uses its logical and communicative skills to help medical students in the proper approach to medical diagnosis and to evaluate their progress in this area. The mechanism for each of these features is discussed. MEDCAT is coded in APL and implemented on a 68000-based microcomputer.

INTRODUCTION

Patient populations on medical floors of teaching hospitals are heavily skewed toward the seriously ill and the previously diagnosed. As a result, the variety of diagnostic problems encountered is limited. The reasons for this are not relevant here, but one of the generally recognized consequences is that medical students do not have enough opportunity to develop their clinical reasoning. The primary purpose of MEDCAT is to help fill this void by providing students with the opportunity to sharpen their diagnostic skills through interaction with a computer program that contains a carefully selected range of patient records.

It was necessary to give MEDCAT four specific capabilities to achieve its goal. First, MEDCAT arrives at its own diagnoses by using the empirical data contained in a patient's medical record and its knowledge of medical logic. Second, it can explain any step in its reasoning in response to questions. Next, it chooses appropriate strategies and takes the initiative in asking questions as it guides and evaluates the student's performance. Finally, it adds to its own knowledge and reasoning ability by free-format discussions with expert physicians.

The essence of MEDCAT's intelligent behavior is found in two areas: the method used to represent both its medical knowledge and logic as data external to the code, and the way these data are interfaced with the external world through its natural language abilities.

Because we wish to illustrate not only what the program's function is, but also how it works, we must use some medical terms and concepts that may be unfamiliar. However, we simplify the diagnostic logic to illuminate the explanation of the process itself. MEDCAT is coded in APL and implemented on a 68000-based microcomputer.

REPRESENTING MEDICAL KNOWLEDGE AND LOGIC

The data in the program are viewed as a network of nodes and pointers. Figure 1 shows 8 nodes (boxes and circles) and the 11 pointers (arrows) that connect them. These pointers may be either positive (facilitory) or negative (inhibitory). Facilitory pointers are shown as solid arrows; inhibitory pointers are dashed arrows. Each pointer is associated with a numeric value.

Node Descriptors

Each node has a descriptor (noun phrase), which is used by the program to identify the subject of a question and to generate its response. These node descriptors are shown in abbreviated form in Figure 1. The complete node descriptors, together with their node numbers, are as follows:

156 IGM antibody to hepatitis-A virus (IGM AB–HAV)
157 IGM antibody to hepatitis-A virus (IGM AB–HAV)
158 Hepatitis-A (acute)
165 Hepatitis-B surface antigen (HBSAG)
166 Hepatitis-B surface antigen (HBSAG)
167 Hepatitis-B (acute)
168 Hepatitis non-A non-B (acute)
345 Viral hepatitis (acute)

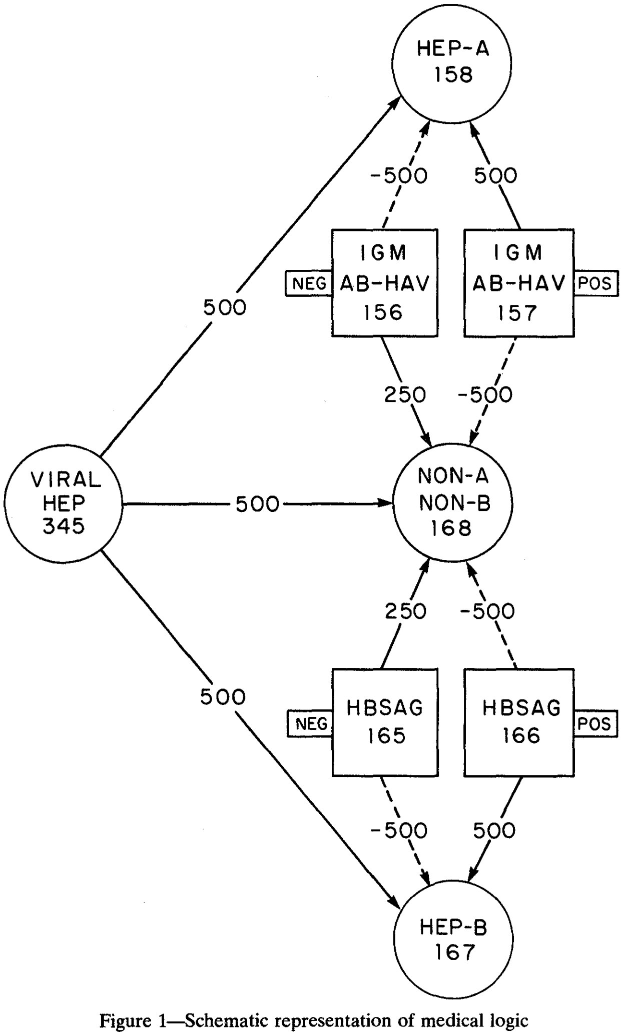


Figure 1—Schematic representation of medical logic

TABLE I—Numeric attributes

	156	157	158	165	166	167	168	345
Profile	6	6	11	6	6	11	11	10
Adjective	2	3	6	2	3	6	6	6
Threshold	100	100	500	100	100	500	500	500
Patient record	NEG	POS		NEG	POS			
Stimulus	1	0	0	1	0	0	0	0

Numeric Attributes

Each node also has five additional attributes, each of which is stored as a vector (Table I). The node **profile** is an integer indicating what the node represents. The first six represent empirical data (1, chief complaint; 2, demographic data; 3, history; 4, symptoms; 5, signs; and 6, laboratory tests). Profile types 7–11 represent diagnoses with varying levels of specificity—11 is the most specific. Nodes 156, 157, 165, and 166 are laboratory tests. The remaining are diagnostic nodes. The three with profiles of 11 are more specific than viral hepatitis (10)—they are specific forms of viral hepatitis.

Figure 1 shows four diagnostic nodes as circles, together with the node numbers. The four empiric nodes are shown as boxes. The significance of a positive hepatitis-B surface antigen node, and thus the group of pointers emanating from it, differs from that of a negative one. As a result, empiric nodes occur in sets of 2 to 11. Only one node in such a set may be on. The HBSAG, for example, may be positive or negative, or it may not have been tested. It may not be both positive and negative. In the patient we will discuss, the IGM AB–HAV and the HBSAG are both negative. Each diagnostic node, on the other hand, is unique; for instance, there is only one node with the descriptor "hepatitis-B."

The **adjective** value of a node is an integer used to index a list of adjectives. This permits the program to generate responses containing phrases such as "the *positive* hepatitis-B surface antigen" or "the *markedly elevated* white blood cell count."

The **threshold** of a node is a positive integer. If the algebraic summation of all the active afferent (incoming) pointers to a node equals or exceeds the threshold, the node is "fired," and its efferent (outgoing) pointers become active.

The **patient record** contains the actual data for empiric nodes. This may be a quantitative value for variables such as age, temperature, total serum bilirubin, or a qualitative adjective as in the examples shown here (nodes 156, 157, 165, and 166). Diagnostic nodes have place-holders, but no values, in the patient record.

A physician postulates hypotheses and diagnoses from the empirical data available. Diagnostic nodes, therefore, must be turned on by execution of the program. Whether or not an empiric node (profile types 1–6) is on, however, is determined solely by the patient record. The **stimulus** is a binary vector where 1 indicates that the node is on, and 0 indicates that it is off. Diagnostic nodes have stimulus values of 0. The threshold, stimulus, and patient record are patient specific, and contents vary from patient to patient. The thresholds are calculated by the program.

TABLE II—Pointer matrix

From	156	156	157	157	165	165	166	166	345	345	345
Value	−500	250	500	−500	−500	250	500	−500	500	500	500
To	158	168	158	168	167	168	167	168	158	167	168

Pointer Matrix

The necessary information about each pointer is stored in a column of a three-row matrix. Row 1 indicates the node from which the pointer originates and row 3 indicates the node to which it points. Row 2 contains the value of the pointer. That part of the pointer matrix corresponding to Figure 1 is shown in Table II.

CONSULTATION MODE

The physician or student selects a patient. The program retrieves the patient record and executes the diagnostic subroutine. This takes about 3 seconds. The physician may then ask the computer questions about the reasoning it used to evaluate the patient's problem:

Physician: What was your diagnosis?
Computer: Hepatitis non-A non-B.

Figure 1 shows the final phases of the diagnostic process in which the program chose hepatitis non-A non-B. Node 345 (viral hepatitis) was activated during earlier iterations because the algebraic summation of its afferent pointers (not shown) exceeded its threshold. Nodes 156 and 165 were turned on by the stimulus. The algebraic summation of the active pointers from these three nodes to nodes 158 and 167 was 0. The summation at node 168 (hepatitis non-A non-B) was 1000, compared with its threshold of 500.

The physician then queries the program about its reasoning:

Physician: Why?
Computer: Because the negative IGM antibody to hepatitis-A virus and hepatitis-B surface antigen and the diagnosis of viral hepatitis confirmed hepatitis non-A non-B.

We will put aside for the moment the question of how the program knew which node was being asked about and focus only on how it applied the available data (logic) to answer the question. The "why" nature of the question caused it to look at the facilitory pointers to hepatitis non-A non-B and to enclose the node descriptors for the three nodes involved in appropriate narrative text. The word "negative" was determined from the adjective vector. The verb "confirmed" was dictated by the magnitude by which the algebraic summation exceeded the threshold.

Physician: Did you also think of hepatitis-A?
Computer: Yes, hepatitis-A was considered, but rejected.
Physician: What made you reject it?
Computer: Because the negative IGM antibody to hepatitis-A virus argued strongly against hepatitis-A.

The program says that hepatitis-A was considered because there is at least one active facilitory afferent pointer going to it. It says hepatitis-A was rejected because the node was not turned on. The reason for rejection was obtained by looking at the active inhibitory pointer.

Physician: Why the IGM AB–HAV?
Computer: To determine the likelihood of hepatitis-A and hepatitis non-A non-B.

When the "why" question refers to an empiric node, the program looks to the efferent pointers from the node to find out what decision (diagnostic node) these pointers would influence.

STUDENT MODE

When medical students use MEDCAT to practice their diagnostic skills, they experience three phases of the program in sequence.

Phase I

Aside from a brief narrative summary generated by the program, the student obtains patient information only by asking questions. The program keeps track of these questions and will use this information later to guide its questioning of the student's reasoning.

When a student tries to obtain demographic, historic, or symptomatic information, he assumes he is speaking directly to the patient. This permits the program to evaluate and comment on interviewing skills. A common error of beginning students is to ask questions that influence a patient's answer.

Student: Did you have pain below the ribs on the right side?

At this point the program will break in with the following comment and require the student to ask another question:

Computer: That is a leading question.
Student: Have you had any pain?
Patient: Yes.
Student: Where was the pain?

If the question the student asks the patient has a syntax similar to "did you have" or "was there," and if it matches only one node in the list of node descriptors with profile types 2–4, it is treated as a leading question. If it matches on more than one, it is accepted. There is only one node descriptor that contains all the words "below pain ribs right side." There are many that contain only "pain." The patient gives an affirmative answer if it is true for any of the matches—otherwise the patient answers negatively. Questions with syntax such as "where was the pain?" or "what was its quality?" are not considered leading questions.

The student also has the option of asking ELIZA-like questions.[1] This is considered good interviewing technique:

Student: Tell me more about the pain.
Patient: It was right under my ribs on my right and occurred after eating.

When a student wants information about physical findings or laboratory data, he assumes he is speaking to the computer. After determining what node the student's question refers to, the program gets its answer from the corresponding value in the patient record.

Student: What were the total serum bilirubin, SGPT, and liver biopsy?
Computer: Liver biopsy was not done.
SGPT was: 750 U/L.
Total serum bilirubin was: 15 MG/DL.
Why did you want the liver biopsy?
Student: Because I was considering malignancy.

When the student asks for information that is not available in the patient record, the program may ask for justification of the request. Whether or not this occurs is determined by the appropriateness of the request, the expense of the test, and the risks involved.

Phase II

When the student finishes asking for patient data, the program takes the initiative. By this time it has made its own evaluation of the patient's problem. It also knows what questions the student asked and, therefore, what conclusions the student may or may not draw. The program chooses a strategy based on these two sets of data. The program begins by asking about the diagnosis:

Computer: What was your diagnosis?
Student: Hepatitis non-A non-B.

If the student's answer agrees with the program's own evaluation, as happens here, the program determines whether the reasoning was sound:

Computer: Why did you diagnose hepatitis non-A non-B?
Student: Because the IGM AB–HAV and HBSAG were negative.
Computer: How do you know he had liver disease?
Student: Because there was evidence of acute hepatitis.
Computer: How did you rule out toxic hepatitis?
Student: There was no history of exposure to toxins or drugs.

IGM AB–HAV and HBSAG are recognized abbreviations for IGM antibody to hepatitis-A virus and hepatitis-B surface antigen, but so far the student's reasoning is incomplete. We would want to know whether viral hepatitis was part of his logic. Viral hepatitis and acute hepatitis are not synonymous. Acute viral hepatitis is one form of acute hepatitis and is, therefore, more specific (profile type 10 vs. type 9). However, we do not want to teach students to parrot answers. This student is doing fine. The program assures itself that the student is on the right track, without mentioning viral hepatitis, by asking why he rejected another type of acute hepatitis. The program will continue asking questions such as what other diagnoses were considered, why they were considered, and why they were rejected. It keeps a running score of the student's performance, and when the student is doing this well, it terminates the discussion after three or four major topics have been explored.

Other strategies are dictated by the student's answers to the initial and subsequent questions. First, if the student gets the right diagnosis, but for the wrong reasons, the program pursues this vigorously. Next, if the student were to give hepatitis as the diagnosis, the program would ask questions designed to lead up to the more specific (etiologic) diagnosis. Finally, if the student does not have sufficient knowledge of the patient's data because the questions were not asked in Phase I, these findings are presented to the student at the appropriate time.

Phase III

During Phase I the students are allowed to ask for empirical data only. They may not ask questions about diagnoses. The program asks the questions during Phase II. Phase III permits the students to resolve any questions that may have arisen or that were unanswered in previous phases. It is identical in format to the consultation mode.

LEARNING MODE

New knowledge and logic can be added to the program by free-format discussions with qualified experts. In the following example the program has made an erroneous diagnosis. It must correct itself, learn about a new disease, and find the basis for a new diagnosis. This further illustrates the usefulness of the system's method of data representation and its language capabilities.

Physician: What was your diagnosis?
Computer: Hepatitis non-A non-B.
Physician: That's wrong.
Computer: Why is it wrong?
Physician: Because the positive beef red cell absorption test excluded it.
Computer: Why did you order the beef red cell absorption test?
Physician: Because the heterophil agglutinin test was positive.
Computer: Why did you order the heterophil agglutinin test?
Physician: Because there were enlarged cervical lymph nodes and a flu-like syndrome.

The data nodes referred to in this discussion already exist. The program is simply adding pointers to the pointer matrix (Table II). The physician's answer to why it was wrong caused a new (inhibitory) pointer to be added from an existing node (38), called "beef red cell absorption," to "hepatitis non-A non-B" (168). The program chose node 38 over node 37, which has the same descriptor, because 38 has a stimulus value of 1. Verification is done by comparing its adjective value with the adjective used by the physician. The verb "excludes" causes the pointer to be sufficiently negative to outweigh all facilitory pointers. This is accomplished by adding a column (38, −1100, 168) to the pointer matrix.

The reason given for ordering the beef absorption test added a pointer with a value of 1. Such weak pointers to empiric nodes affect only the conversational phases of the program. These are discussed when we describe how the threshold is calculated.

The program now understands why its diagnosis was incorrect, but it does not know what the diagnosis should be. It asks:

Computer: What is the diagnosis?
Physician: Infectious mononucleosis.
Computer: Why did you diagnose infectious mononucleosis?
Physician: Because the beef absorption and heterophil agglutinin tests were positive.
Computer: Do you mean beef red cell absorption test?
Physician: Yes.
Computer: Were there any other reasons?
Physician: Enlarged cervical lymph nodes and flu-like syndrome.
Computer: Any others?
Physician: No.
Computer: How important (on a scale of 1–5) is the beef red cell absorption test?
Physician: 5.

Infectious mononucleosis represents a new node that is added to the list of node descriptors. It is a diagnostic node and, because it is replacing hepatitis non-A non-B, it is assumed to have the same profile type. Because it is a diagnostic node, place-holders are required only for its adjective type, stimulus value, and patient record. Its threshold, as with any node, is calculated prior to execution.

Four pointers to infectious mononucleosis from the four nodes indicated must be added. Because the physician did not use any quantitative verbs or adverb–verb combinations to indicate pointer values, these are explicitly requested. This process is repeated for the remaining pointers. To test the results of this learning session, the diagnostic subroutine must be executed again:

Physician: What was your diagnosis?
Computer: Infectious mononucleosis.
Physician: Why?
Computer: Because the enlarged cervical lymph nodes, the positive beef red cell absorption and heterophil agglutinin tests, and the diagnosis of flu-like syndrome confirmed infectious mononucleosis.
Physician: Why did you order the beef absorption?
Computer: Because of the possibility of infectious mononucleosis, and because the heterophil agglutinin test was positive.
Physician: Why did you reject non-A non-B?
Computer: Because the positive beef red cell absorption test excludes it.

Evaluating the Logic

The program checks for obvious inconsistencies. It would reject "because the heterophil test was negative" because this disagrees with the patient record. Aside from conflicts with empirical data, however, the program will accept any line of reasoning as long as it can be made consistent with what it already knows. The program stops asking questions and accepts the logic when afferent pointers exist to each new diagnostic node.

Calculating the Threshold

The thresholds of new nodes need not be set by the physician. The threshold of all empiric nodes is set arbitrarily at 100. The values of afferent pointers to empiric nodes can be either −1 or +1. The algebraic sum of such pointers never exceeds the threshold of the empiric node and therefore cannot turn it on. Whether or not an empiric node is activated is determined solely by the patient data. This is an important theoretical point. Certainly one would not want the answer to a question or a test to be affirmative simply because the question was asked or the test was done.

These weak pointers are used when the program is generating answers to questions and when it evaluates a student's reasoning. The beef red cell absorption test was ordered because the heterophil agglutinin test was positive. As indicated earlier, we are particularly concerned that the student have adequate reason for requesting laboratory tests. When requesting information from experts, the program asks for reasons for laboratory tests, although it does not ask for reasons for other types of empiric nodes.

The program calculates the thresholds for diagnostic nodes. The basic process consists of dividing the sum of positive afferent pointers to the node by two and using this quotient or 500, whichever is greater, as the threshold. The sum is divided by two because we want to mimic varying degrees of certainty in a diagnosis. When the algebraic sum of afferent pointers to a node equals or barely exceeds the threshold, this is interpreted as a tentative diagnosis or working hypothesis. If it is more than twice the threshold, the diagnosis is confirmed.

Only pointers from active empiric nodes are used. This is necessary because each node in the set that represents a test has a different pointer value. One would not want to add these together when only one of them can be relevant. The use of pointers from active empiric nodes also is desirable because it permits the flexibility one sees in observing physicians at work. If a specific test or piece of information is not available, a given diagnosis should not be ruled out. A positive beef absorption test may confirm infectious mononucleosis, but the presumptive diagnosis can be made without it.

Afferent pointers from diagnostic nodes are counted, whether active or not, because there is no way of determining this before executing the diagnostic subroutine. This is consistent with the role intermediate diagnoses should play. If an intermediate diagnosis is missing, the more specific diagnosis often should not be made.

NATURAL LANGUAGE PROCESSING

On the input side the basic tasks the program faces are to determine the purpose of the question and the nodes that are being referred to and (particularly in the learning mode) to isolate certain adjectives and verbs with quantitative implications. This must be done within a reasonable response time (2–3 seconds). Each task is simplified because of the limited verbal domain in which we work.

Word Types and Parsing

The sentence is first parsed into functional units or phrases. This is done by checking each word against a list of 128 keywords; each keyword is associated with a numeric value. Thus, a numeric vector is established with a value (word type) for each word in the sentence. These numbers indicate several things, including whether the associated word (or punctuation) should start or end a phrase; whether it is a recognized adjective, verb, or profile word; and whether it is negative or positive. "Excludes" encodes to −6.9. The 6 indicates that it is a verb that begins a phrase unless preceded by another verb or adverb. The minus sign indicates that it requires a negative pointer. The .9, together with its sign, indicates that the negative pointer must be strong enough to prevent firing of the node.

The program breaks up the sentence fragment, "because the positive beef red cell absorption test excludes it," into four separate phrases:

1. *Because*
2. *The positive* beef red cell absorption *test*
3. *Excludes*
4. *It*

The words in the keyword list are italicized for illustrative purposes. "The" is one of the words that flags the start of a noun phrase. A verb such as "excludes," unless preceded by an adverb or helping verb, signals the beginning (and end) of a verb phrase. The remaining italicized words are not needed for parsing in this instance.

It is important that the keyword list contain no nouns except profile words. The program uses the first word to characterize a phrase. If this word happens to be an article, as above, the phrase is clearly a noun phrase. However, articles often are omitted by students, particularly when they ask for lists of laboratory results. Commas, verbs, pronouns, and conjunctions signal the end of one phrase and the start of another. Because the keyword list contains most of the non-noun words that are used in our limited domain, the program may assume that a phrase is a noun phrase if it does not begin with a keyword.

Node Selection

The list of descriptors is treated as a string and is searched for the occurrence of single words in the input sentence. One purpose of parsing is to eliminate unnecessary words from the search. Not only does this narrow the.process to the noun phrase, but the italicized words in the example phrase are deleted before searching begins. This preprocessing maintains the necessary response time even though the descriptor list contains 30,000 characters and is growing.

In the consultation and student modes, the program picks the node with the highest number of matching words. In a previous example, "beef absorption" matched "beef red cell absorption." This is sufficient for a presumptive match, but because new nodes and pointers are being added, the program asks for verification in the learning mode: "Did you mean beef red cell absorption?"

In the string search, input words are preceded but not followed by a space. For this reason, "nucleosis" will not match with "mononucleosis," but "mono" will. This permits users to follow the inclination to abbreviate words. If there is no match, the program truncates each input word to five characters and repeats the search. This corrects for many typing errors. Synonyms and additional descriptive words are included within parentheses in the node descriptor list. These are deleted when the program uses a descriptor in its output, but provides greater flexibility on the input side.

Pronoun Reference

The program always stores the first node found in a sentence. When the program encounters a pronoun, it then substitutes the most recently stored node. This not only reduces the amount of typing, but it also reduces processing time. In the example we are using, "hepatitis non-A non-B" becomes substituted for "it." If the physician were to use a pronoun next time, "beef red cell absorption" would be substituted. When the question contains neither a subject nor a pronoun, as in "why?" a pronoun is assumed and the stored node is used.

Quantifiers

When the program is adding new nodes, it looks for words ("test," "sign," or "symptom," for example) that indicate the profile type to be assigned. Certain adjectives, adverbs, and verbs are used to assign pointer values. If such words are not found, the program asks for the missing information. In consultation mode the program also uses adjectives and verbs to determine the form of its answer:

Student: Was the HBSAG positive?
Computer: No, it was negative.

Generating Output

The program uses interrogative words (why, how, did, was, what, and where) to branch to subroutines that handle each type of question. Combinations of these words are also used to increase specificity. "How did," "how was," and "what made," for example, are treated as "why" questions. Other instances of "how" are treated as "what" questions. Different tenses of the same verb are equated.

Student: How high was the temperature?
Computer: 101 F.

Subroutines called "was" and "did" handle true-or-false questions. "What was," except for "what was your diagnosis?" involves retrieval of data from the patient record. "Why" questions cause the program to examine the afferent or efferent pointers to or from the node that is being asked about in four ways:

1. "Why did you diagnose infectious mono?" calls the active afferent facilitory pointers.
2. "Why did you reject non-A non-B?" depends on the active afferent inhibitory pointers.
3. "What did the positive heterophil make you think of?" depends on the active efferent pointers from the node.
4. Because it is an empiric node, "Why did you order the beef absorption?" involves efferent pointers from **beef absorption**, and afferent pointers to it.

Enclosing these data (nodes, pointers, adjectives, profile) in appropriate narrative prose is simply a matter of applying flexible templates. The magnitude by which the algebraic sum exceeds the threshold determines the choice of verb phrases ("confirms," "strongly suggests," "suggests," or "is consistent with").

DISCUSSION

MEDCAT has certain features in common with MYCIN.[2] It has an instructional mode, it exhibits knowledge acquisition, and it can display reasons for its decisions. However, the single feature that most clearly distinguishes MEDCAT from MYCIN and other rule-based artificial intelligence systems is the representation of both its knowledge and its logic as a numeric matrix that serves a variety of functions. MEDCAT uses the matrix to arrive at its diagnoses, and the simple parser and text generator permit the program to interface with the user. This representation also accounts for the ease with which knowledge and logic can be evaluated and changed during free-format discussions. APL, which is often described as an array-processing language, is well suited to this purpose.

The learning that MEDCAT exhibits addresses the general problem of getting the most qualified experts to contribute to the logic and knowledge base. Experts usually are unaccustomed to analyzing the essential mechanisms that may underly their particular intellectual skills. They are experienced, however, in making decisions and justifying them once they are made. This is exactly what MEDCAT does—it engages experts on familiar intellectual turf. By asking questions such as "Why do you disagree?" or "Why do you think that?" or "Why did you order that test?" the program requires no more of physicians than what is expected of them on their teaching rounds.

ACKNOWLEDGMENTS

This work was supported in part by grants from the Frances L. and Edwin L. Cummings Memorial Fund and the Anonymous Gift Fund of Cornell University Medical College.

REFERENCES

1. Weizenbaum, J. "ELIZA—A Computer Program for the Study of Natural Language Communication between Man and Machine." *Communications of the ACM,* 9 (1966), pp. 36–45.
2. Davis, R. "Consultation, Knowledge Acquisition, and Instruction: A Case Study." In P. Szolovits (ed.), *Artificial Intelligence in Medicine.* Boulder, Colo.: Westview Press, 1982.

PRISM: PRototype Inference SysteM

by P. HIRSCH, M. MEIER, S. SNYDER, R. STILLMAN
IBM Corporation
Palo Alto, California

ABSTRACT

A research project at IBM's Palo Alto Scientific Center, called PRISM (PRototype Inference SysteM), is an experimental "shell"—a general purpose system. This system is designed to provide a base for the construction of a variety of specific expert systems. PRISM is in operation within IBM at several sites where it is being used for building applications. A knowledge base builder uses the English-like rules and parameters to construct an application more rapidly than can be done with standard programming techniques.

INTRODUCTION

A research project at IBM's Palo Alto Scientific Center, called PRISM (PRototype Inference SysteM), is an experimental "shell"—a general purpose system. This system is designed to provide a base for the construction of a variety of specific expert systems. The motivation for designing PRISM arose from observing the difficulty with which expert systems are built—often, each system is one of a kind. There is a need for an easy way to build expert systems that can run efficiently and be used in a variety of problem areas.

There are three users of an expert systems shell:

1. The client or end-user of the application
2. The expert
3. The knowledge engineer, who helps the experts encode their knowledge

PRISM incorporates a set of procedures used to design a variety of applications—an empty system into which the knowledge-base builder (an expert or knowledge engineer) inserts his own rules (to define the knowledge base) and choose a reasoning method (the inference engine) that will apply the rules. The result is a specific export system ready for the end-user. PRISM incorporates an editor program that allows a knowledge engineer to create the knowledge base in the form of English-like rules, parameters, and controls.

PRISM also includes a set of basic inference-engine-processing functions. Examples include backward chaining (obtaining a value for a desired goal by working backward to a given premise) and forward chaining (proceeding from given data to make inferences and draw conclusions or carry out actions). From these basic functions, PRISM's control language permits the building of more complex functions. It is the application of these functions to the rules in the knowledge base that determines the particular characteristics of the expert system. In each system the base set of knowledge-processing functions is the same; what differs is the sequence and focus of their execution.

The PRISM system is now in operation on an IBM System/370 under the VM/CMS operating system. A knowledge base builder uses the English-like rules and parameters to construct an application more rapidly than is possible with standard programming techniques. A knowledge engineer can use either of two standard inference techniques—backward or forward chaining—and can control the inference process through a control language. In addition, a knowledge engineer can organize or group the knowledge base into hierarchical structures, called focus control blocks (FCBs). Each FCB has its own inference engine and its own control steps. PRISM also allows for easy attachment of external procedures to the system to provide or use information from other programs, files, or data bases.

KNOWLEDGE BASE OBJECTS

A PRISM knowledge base is constructed from three types of objects: rules, parameters, and FCBs. Rules contain the logic structure and provide heuristic and definite facts about the application. Parameters are the basic value-holding objects of the system and also provide initial value constraints. FCBs allow a knowledge base builder to control PRISM through a control language and also subdivide the knowledge base into components. All of the objects are entered through an integrated editor that automatically parses and analyzes the information as it is entered. If there is a semantic or syntactic error, immediate feedback is given that allows the knowledge base builder to correct the error at once.

RULES

The production rules are of the "IF... THEN... " form, for example:

1. If income > 30000 then tax _ rate > .3
2. Fif Animal _ produces is 'gives milk' then animal _ class is 'mammal'
3. Fif animal _ class is 'insect' or animal _ eats is 'nibbles at cotton' then there is some evidence that animal is 'boll weevil'
4. Fif mammal _ class is 'ungulate' and certainty that (animal is 'zebra') < .5 then there is certainty .8 evidence that animal is 'horse'

Note that Rules 2 through 4 are "fuzzy" if rules, FIF rules, which propagate the certainty of the premise to the certainty of the conclusion, whereas Rule 1 is an IF rule and the certainty of the premise is not propagated to the uncertainty of the conclusion. In Rule 1, if the rule premise is true, the conclusion will always be asserted with certainty 1.0. Also note that Rule 4 is a self-referencing rule; that is, the parameter ANIMAL is mentioned both in the premise and in the conclusion.

CERTAINTY

PRISM can maintain heuristic knowledge as well as factual knowledge through the use of uncertain reasoning. A knowledge base builder can create fuzzy rules, such as the FIF rules, through phrases like those shown in rule 3, "there is some evidence," and in Rule 4, "certainty that (animal is 'zebra')

< .5" and "certainty .8 evidence that." In addition, the user can supply a certainty value along with the answer, for example:

What is the animal class?
<u>.8</u> Insect
___ Mammal
___ Ungulate

In this example, the user determined that the animal class was insect, with certainty .8.

PARAMETERS

In addition to rules, the knowledge-base builder creates parameters. These parameters can have properties such as constraints, prompt messages, and indications of where information is to be obtained. The parameter constraint could be a list of values or a range of values. For example, the parameter COLOR might be taken from the list ('red', 'white,' 'blue') or the parameter INCOME could be constrained to be >0. Parameter values are checked against these constraints during knowledge-base building and consultation. Each parameter has a property that indicates whether the value for that parameter is to come from the processing of rules, from the end-user, from external procedures, or from a default value. In addition, the knowledge-base builder can specify the sequence that the system should use in attempting to obtain the value of the parameter.

For example, the knowledge-base builder might specify that the system should attempt to obtain the parameter value according to the following sourcing sequence:

External data
Rule consequent
User will input from terminal
Default will be taken

CONTROL

The knowledge engineer may wish to control the dialogue and the processing of the rules in an expert system. The PRISM system permits this through a knowledge object called the focus control block (FCB). The knowledge engineer can partition the knowledge-base hierarchically using FCBs. For example, a system configurator might want the expert system first to focus on the applications, then on the CPU required, and finally on peripheral hardware, such as tape or disk drives. Each component is in a separate control block and they are hierarchically related to one another.

Each FCB can have its own control language. As an example a set of control primitives might be as follows:

Ask initial data;
Determine goals
—Order rules by least unknown premises first;
Display results;

In this example, initial data, goals, and results each constitute a group of parameters that the knowledge-base builder has provided. In this example the PRISM system initially would ask for the parameter values that are listed in the group initial data. It then uses a backward-chaining algorithm to determine the goals (another list of parameters) of the system. Upon reaching the goals, the system displays the parameters indicated by the results, which in many cases contain the initial data and the goal parameters.

APPLICATIONS

PRISM is currently being used at several IBM sites. These applications range from diagnostic applications to planning and advisory applications. A sampling of the applications includes a contract advisor for joint research contracts, a system to configure hardware, a hardware diagnostic system, and a system to verify information going into a database.

The impact of AI technology on VLSI design

by ROBERT S. KIRK
Gould AMI Semiconductors
Twain Harte, California

ABSTRACT

Twenty years ago, artificial intelligence technology promised to revolutionize the world. As time would tell, advances in artificial intelligence have taken significantly longer than expected. Slow progress created skepticism and disinterest in the technology. Today there is a great deal of renewed interest in the field, tempered by the slow progress of the past twenty years. This new interest is focused on domain specific artificial intelligence applications, rather than the broad problem solving capabilities originally proposed. In addition this interest is focused on domains offering exceptional return on investment, either through direct profits or through leveraging of scarce resources. This paper surveys the potential impact of artificial intelligence technology on the VLSI design domain. This domain is characterized by a fifteen year evolution of computer aided design tools, a chronic shortage of skilled integrated circuit designers and ever growing demands for shorter design spans, reduced costs and design error rates.

INTRODUCTION

Advances in integrated circuit (IC) fabrication technology are rapidly outpacing IC design capabilities. The first ICs contained small scale integrated (SSI) functions, which were straightforward to design. As device counts increased, early design methods quickly became obsolete. Today computerized tools are widely used to design very large scale integrated (VLSI) circuits. These software tools are themselves very complex. Yet with advances in IC fabrication technology, the pressure to solve more difficult design automation problems continues to increase. Present software technology has been pressed to the practical limit in the current generation of VLSI design tools. Entirely new approaches to the VLSI design problem are required.[1,2] Artificial intelligence (AI) technology may hold the key to solving this problem.

This paper explores the potential contributions of AI technology to VLSI design and attempts to answer the following questions. Will AI offer truely useful solutions, or will it go the way of the past twenty years? What significant changes in VLSI design can be expected in the next three, five or ten years due to AI?

The degree to which AI impacts VLSI design will significantly affect the entire computing community. Advances in VLSI supports advances in computing hardware and together they feed advances in AI research.

Before describing how AI technology will be used to advance VLSI design capabilities, VLSI design requirements are reviewed, followed by an overview of existing AI based VLSI design tools. Then the salient features of AI technology are examined to draw some conclusions on their impact on VLSI design tools and methodology.

TRENDS IN VLSI DESIGN

The VLSI design tool domain is commonly referred to as Computer-Aided Design (CAD) or Design Automation (DA) as it applies to integrated circuit design. The complexity of ICs is increasing so rapidly that ICs are no longer limited to simple logic devices. Rather complete digital systems are being designed on a single silicon chip.[3] Thus the term VLSI design means both digital systems design and IC design.

Digitizing

In the early days, CAD tools were developed to automate the tedious and error-prone task of creating IC photomask artwork. By digitizing the photomask drawings or layouts, a computerized editing system could be used to make changes. New artwork was then generated automatically on a photo plotter. A digitizing system is the graphical analogy to a textual word processing system. The main drawback of digitizing systems is that they did little to help a person perform the design task.

Checking and Analysis

Next came a generation of checking and analysis tools. These tools are similar to a spelling checker in a word processing system. Their purpose was to automate the incredibly difficult and tedious task of checking a layout for design rule violations or analyzing the performance of an electronic circuit. These tools helped the designer immensely by eliminating a great deal of mechanical work. A person, however, still had to perform all of the creative design work.

Because these CAD tools were primarily mechanical in nature, they lent themselves to implementation by algorithms that were not unreasonably complicated. This is not to say that these algorithms were trivial, only that AI techniques were not required.

Test and Diagnosis

At roughly the same time that checking and analysis tools were under development, interest in the test and diagnosis area increased. Procedures for testing and diagnosing problems in SSI complexity ICs were totally inadequate for testing complete VLSI systems on a chip. Numerous algorithmic approaches were tried, but there has been relatively little success to date. Researchers are now turning to AI techniques to see if this difficult problem can be solved.

Synthesis

With reasonable performance from checking and analysis tools, research attention shifted to the general area of design synthesis and is now the most active area in the VLSI design tool domain. The idea behind synthesis is to solve the problem of automating the design process. Most synthesis systems operate in a series of steps known as decomposition and refinement as illustrated in Figure 1. Within the area of synthesis, there are two major topics: silicon compilation, and automatic test generation.

Silicon Compilation

The term silicon compilation was first coined by Johansen[4,5] to describe a process, similar to a software compiler, whereby a textual chip description would be automatically compiled into layout artwork. Today this concept has been widely expanded and now divides into two broad areas: functional specifications to structural netlist compilation, and structural netlist to layout artwork compilation as illustrated in the Y-diagram in Figure 2.

A structural netlist contains the information found in an engineer's logic diagram. Logic symbols are converted into a netlist format usable by the computer. The information is said to be structural because it describes which logic gates will be used and how they are to be interconnected. Information on how it works (function) and details of the chip layout are absent. An example of a logic diagram and netlist are shown in Figure 3.

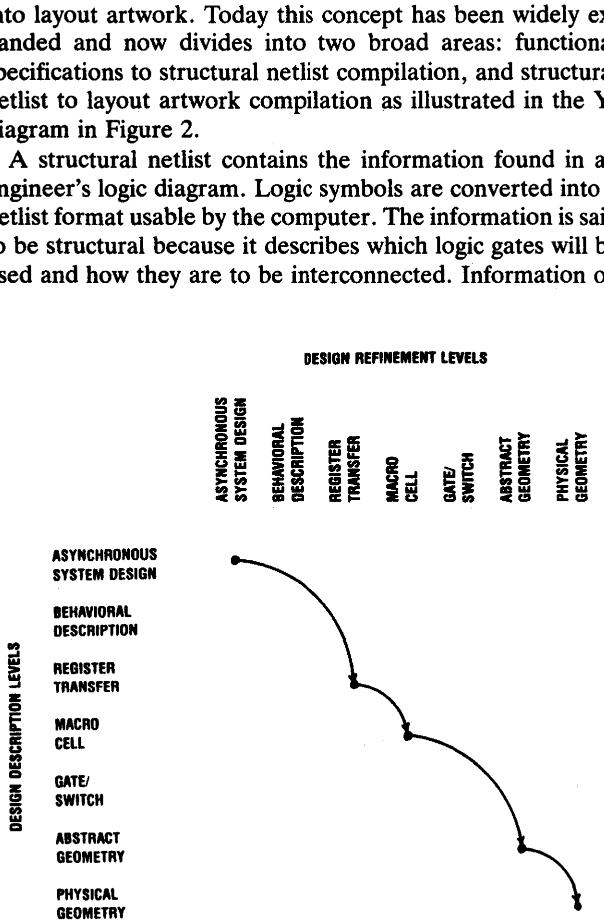


Figure 1—Decomposition and refinement steps

Structural netlist to layout artwork compilers include common tools such as placement and routing for gate array and standard cell chips. These tools automatically perform the design steps involved in deciding where to place cells and how to route the interconnections so as to minimize chip area. Some such tools can actually perform a better job than human designers, at a fraction of the time and with no mistakes.[6]

Another form of layout artwork compiler generates layouts directly from a two dimensional layout language.[7] Layout generators create detailed layout cells for use by a structural netlist to layout artwork compiler. In a sense the cell generator is used to build up the target machine language instructions (cells) of the compiler in terms of a micro code sequence (transistors and connections). Figure 4 shows a simple generator input and output. In some cases, generators are used to create more complex layouts such as a complete datapath for a CPU.[8,9]

Functional specifications to structural netlist compilers perform the task of converting abstract English language descriptions of the system's desired performance into a logic diagram. The first problem is that deciphering English language descriptions is non-trivial. This problem is avoided by inventing a constraining hardware description language (HDL) for writing functional specifications.

The first functional specification compilers performed tasks such as generating programmable logic arrays (PLA). Boolean equations were converted to structural information and then to layout artwork.

Current efforts in this area are much more ambitious. The long term goals are to be able to compile a very high level HDL into logic for any type of digital system. Most research

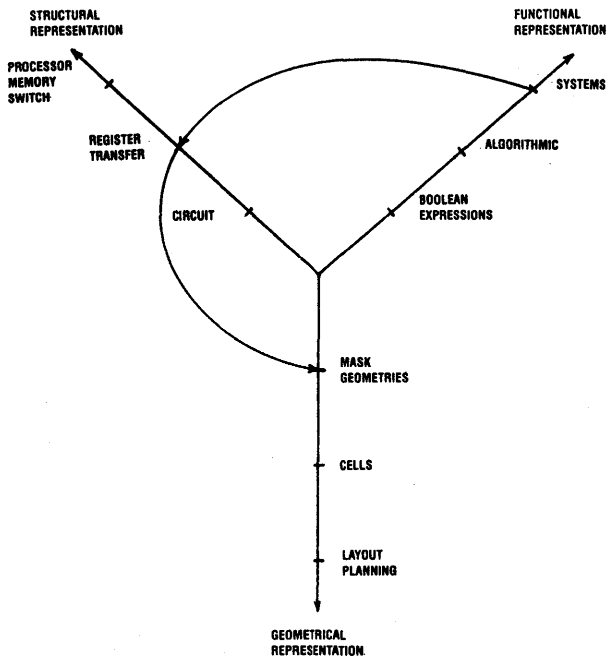
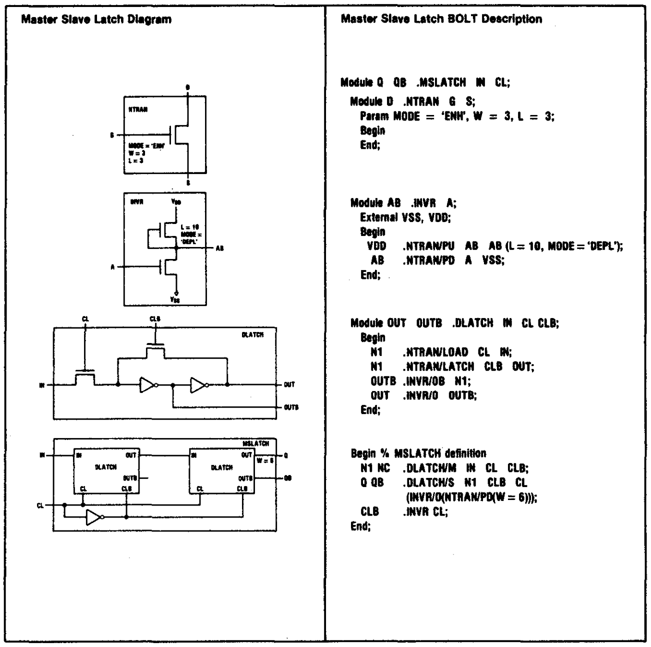


Figure 2—The silicon compilation process



Master Slave Latch BOLT Description

```
Module Q  QB  .MSLATCH  IN  CL;
  Module D  .NTRAN  G  S;
    Param MODE = 'ENH', W = 3, L = 3;
    Begin
    End;

  Module AB  .INVR  A;
    External VSS, VDD;
    Begin
     VDD   .NTRAN/PU  AB  AB (L = 10, MODE = 'DEPL');
      AB   .NTRAN/PD  A  VSS;
    End;

  Module OUT  OUTB  .DLATCH  IN  CL CLB;
    Begin
      N1   .NTRAN/LOAD  CL  IN;
      N1   .NTRAN/LATCH  CLB  OUT;
      OUTB .INVR/OB  N1;
      OUT  .INVR/O  OUTB;
    End;

  Begin % MSLATCH definition
    N1 NC  .DLATCH/M  IN  CL  CLB;
    Q QB   .DLATCH/S  N1  CLB  CL
            (INVR/O(NTRAN/PD(W = 6)));
    CLB    .INVR CL;
  End;
```

Figure 3—Logic diagram and netlist

```
(deflayout inline-inverter
    ((primary-parameters (depl-length 12.0)
                         (enh-length 3.0)
                         (depl-width 3.0)
                         (enh-width 3.0)
                         pull-up-z
                         pull-down-z
                         z))
    (constraints ((c*depl-length pull-up-z depl-width)
                  (c* enh-length pull-down-z enh-width)
                  (c* pull-up-z z pull-down-z))))
(part 'pullup inline-pullup
    (channel-length (>> depl-length))
    (channel-width (>> depl-width)))
(part 'pulldown rect-e-fet
    (channel-length (>> enh-length))
    (channel-width (>> enh-width)))
(align (>> pulldown)
     (>> top-center drain-diffusion pulldown)
     (pt-above (>> diffusion-connection pullup) 1)))
```

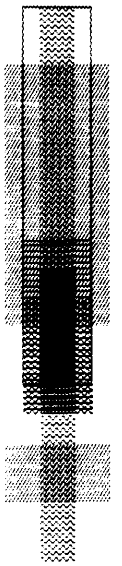


Figure 4—Cell generator description and layout

in this area is employing some sort of AI techniques to achieve this goal.[10]

Automatic Test Generation and Diagnosis

There seems to be a consensus that the goals of automatic test generation and diagnosis require higher level solutions than have been employed thus far. Researchers are realizing that structural netlist descriptions contain far too little information about the "function" of the system.[11,12] With progress in the functional specifications compilation area, it is hoped that breakthroughs will be found in automatic test generation and diagnosis.

OVERVIEW OF EXISTING AI VLSI TOOLS

It is useful to examine the VLSI tools in existence today that use AI technology. Published tools include the XCON expert hardware configuration system, the CMU-DA system, and the MacPitts, Arsenic and Palladio silicon compilers.

XCON was one of the first CAD/CAM tools to use AI technology.[13] XCON performs the difficult task of configuring computer systems for the Digital Equipment Corporation. While XCON is not involved in VLSI design, it demonstrated the use of AI technology in the engineering domain.

The CMU-DA system represents a significant effort to automate the design of digital systems, including CPU and VLSI design. The project has covered many different aspects of the domain. Some software components were written along the lines of conventional CAD tools, while others struck out to experiment with AI technology. These tools, such as TALIB[14] and EMUCS/DAA,[15] use the OPS5[16] production rule system. The CMU projects probably represent the most encompassing efforts to date to explore the use of AI in the systems and VLSI design domain.

TALIB is an expert system for performing the mask layout step starting from a structural netlist. It is effective on small cells with approximately 20 transistors. The production system employs over 1200 rules to construct layouts which are about 10 to 35 percent less area efficient than layouts created by human designers. Cells at this efficiency level are not too useful and the very high number of production rules must have been difficult to collect.

On the other hand the EMUCS/DAA system appears to be more useful. EMUCS and DAA are expert CPU design systems that work at the architectural or functional level. The input to these systems consists of a set of desired machine instructions, and the output is a block diagram and finite state transition table for a CPU. These systems employ only about 70 production rules to obtain acceptable results.

These two systems from Carnegie-Mellon University point out that there are some problems that experts easily solve, and others which the machine can easily solve. In the case of TALIB, the problem is characterized by a relatively small amount of data (20 transistors) and a large number of design rules (at least 1200). For EMUCS/DAA, there is more data (hundreds of machine instructions) and few rules (about 70).

The silicon compiler systems: MacPitts,[17] Arsenic[18] and Palladio,[19] are a bit more conventional because they use an algorithmic approach. The unique quality of these tools however, is that they all employ a search scheme through some abstract design space. Silicon compilers attempt to evaluate a large number of tradeoffs and thereby try out a large number of alternative designs. Figure 5 illustrates how a silicon compiler might evaluate several approaches. This approach differs from the human design approach where only one or two alternative designs are considered. While none of these systems are yet producing competitive layouts as compared to human designs, they have the potential to do so.

USEFUL AI CONCEPTS

Approaching AI from the VLSI design perspective, one would like to extract concepts from AI technology which can be put to practical use. Some concepts are actually not new but rephrased and with the rephrasing often comes new ideas about how to use or implement the concepts.

Computer Languages

Probably the most visible contributions of AI technology are the LISP and PROLOG languages. These languages are

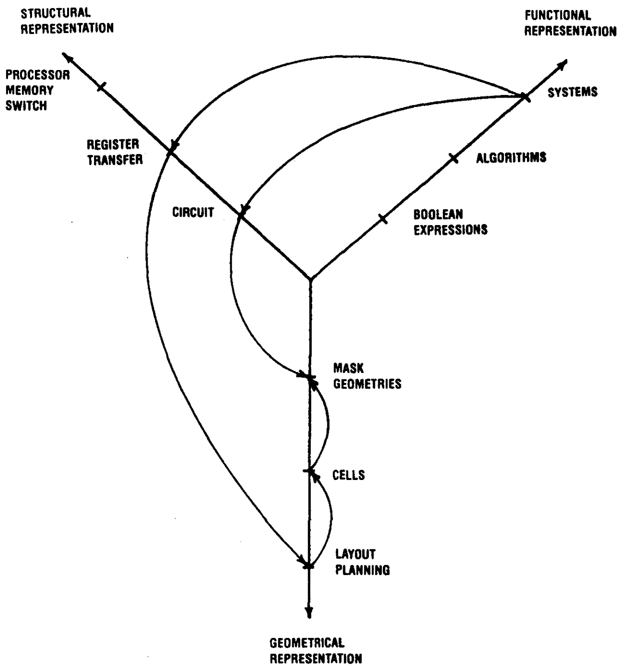


Figure 5—Alternative designs in the design space

significant in the new programming paradigms they each introduce. In addition they allow large algorithms to be implemented in significantly less code. For LISP, a 5 to 1 improvement over PASCAL is common. Since programmers write lines of code at the same rate, independent of the language, this is a significant productivity gain.

Algorithmic vs Production Rule Driven

Many people are taking advantage of the greater inherent capability of LISP to develop more powerful algorithms. These algorithms are generally more symbolic than numerical in nature, much in the same sense that algebra is more powerful than arithmetic. The algorithmic approach is behind some of the advanced VLSI design tools such as layout generators, MacPitts and Arsenic.

Production rule driven systems such as TALIB and Palladio represent a significant departure from the more conventional algorithmic approach. Proponents of each approach strongly believe their approach is correct. From a more objective point of view, it seems reasonable that both approaches are useful, one better than the other in particular cases.

Knowledge Database

Many design tools were originally written in such a way that they embodied the IC fabrication technology in hard coded expressions. For example, a Design Rule Checker (DRC) might check a layout designed in 4-micron NMOS technology. When the technology was changed, say to 4-micron CMOS or to 3-micron NMOS, the entire program would have to be overhauled. It was not long before the IC fabrication technology information was put into a technology file which was read by the program at start up time.

These technology files are a form of knowledge database. The formal concept of knowledge databases, however, introduces new ideas. Many "tricks" embodied in present CAD tool algorithms could be pulled out and kept in a design knowledge database. This would facilitate changes to the tool for handling different design styles.

Expert Systems

Expert systems are loosely defined as a computer program capable of performing tasks at a level equal to or better than experts in the domain of interest. Within this loose definition, a number of conventional VLSI design tools could be considered *expert.* A more proper definition of *expert* systems requires the software to be based on some sort of production rule system. Yet the differences are not as great as they appear. What makes a system perform at an expert level? It usually is the number of IF-THEN conditions in the conventional programming language paradigm and the number of rules in a production system. While the numbers are on different scales, they are a metric of *expertise.*

The benefits of the *expert* systems approach are that fewer rules are required as compared to IF-THEN conditions, and hence the knowledge is more clearly specified. Also, the only code in the system is the rules themselves. This introduces the notion of *granularity* of knowledge. The more granular the knowledge, the easier the system is to modify and extend.

Natural Language

Natural language processing holds the promise of being able to supply the ultimate user-friendly system. A major barrier to the use of VLSI design tools is the user interface. Often the designer must learn a fair amount of "computereze" to deal effectively with the host computer operating system and the individual tools. With natural language processing, a VLSI design tool would be able to deal at a more English like level. This would shorten user training time and avoid mistakes because the tool should be better at "do what I mean" as opposed to "do what I say." The current state of natural language capability supports effective program directed dialogue. The user is asked questions in English and is expected to respond with one word. User directed dialogue capabilities are beginning to emerge with limited capabilities.

A natural language front end would eliminate the need for a hardware description language (HDL) front end to the functional specifications silicon compiler. In removing the rigid constraints of an HDL, the system should do a better job of capturing the sorts of vague and implied tradeoffs and constraints which engineers express in English language functional specifications.

Learning

Machine learning is an extremely attractive idea. A learning system would be able to follow the work of human experts and extract the knowledge for use on similar problems. The main attraction is the dramatic reduction in knowledge acquisition time and cost which is theoretically possible. Unfortunately, research in the machine learning area has not advanced very far at this time.

THE IMPACT OF AI

VLSI design tool developers will assimilate techniques from AI as rapidly as technology and development time permits. AI promises to bring new features and capabilities to VLSI design tools as well as improved design methodologies. It will also have limitations which will keep it from being a panacea.

Promises

Most certainly, AI based tools will make great strides forward in improving user friendliness. Starting with simple things such as program directed dialogue in the near term (three years) and moving toward user directed dialogue (full natural language processing) in the long term (ten years).

The flexibility of production rule systems will enable sophisticated users to modify knowledge databases and alter tools. Because the baggage of implementation details in algorithmic systems is left behind, production rules are fairly easy to understand. The small granularity of production rules makes it more practical for a tool user to understand the IF-THEN rules and judge the impact of modifications. This will enable the user, for the first time, to modify design tools without the assistance of programmers.

With more powerful programming paradigms comes the ability to create more powerful tools. Synthesis tools such as a complete general purpose silicon compiler will emerge in the long term. This tool will leverage scarce engineering resources tremendously and greatly shorten VLSI design times.

Expanded Capabilities

With long term advances in VLSI, computer hardware and AI, it is reasonable to expect performances from VLSI design tools that exceeds human capabilities. This seems quite probable, given the large amounts of data and knowledge required to design ICs. It is reasonable to assume that a machine can eventually do a better job of evaluating complex tradeoffs and selecting the best design from among many design attempts. In addition, an expert design system should be able to complete its task many times faster than a person. Eventually it may be possible to generate working chips from an English description in the time it takes to fabricate the silicon chips.

Limitations

The foregoing probably sounds a bit optimistic and may well be. Natural language understanding is still the subject of much research. Expert systems for problems with small amounts of data and a large number of rules will be developed slowly. Progress towards understanding learning is so slow as to virtually eliminate any chance of using learning to make expert systems acquire rules more quickly, any time in the next ten years.

CONCLUSION

From this brief survey of the VLSI design and AI fields, it is evident that AI technology will significantly alter the way VLSI design is done today. Many human design tasks will be automated, leaving designers to deal with the most difficult and obscure design problems. These advances will pave the way for major revolutions in computing hardware and AI research.

REFERENCES

1. Kirk, R., and T. Daspit. "Making the Design Transition." *Semiconductor International,* 7-5 (1984), pp. 103–107.
2. Kirk, R. "Workstations—A Passing Fad?" *Professional Program Session Record, WESCON/84,* 1984, pp. 1/2.1–1/2.4.
3. Mead, C., and L. Conway. *Introduction to VLSI Systems.* Reading, Mass.: Addison-Wesley, 1980.
4. Johansen, D. "Bristle Blocks: A Silicon Compiler," *IEEE Proceedings of the 16th Design Automation Conference.* New York: IEEE, 1979, pp. 310–313.
5. Ayres, R. *VLSI Silicon Compilation and the Art of Automatic Microchip Design.* Englewood Cliffs, N.J.: Prentice-Hall, 1983.
6. Mehta, S., B. Kirk, M. Ng, and R. Babbar. "CIPAR—A Complete Correct-By-Construction Placement and Routing System," *IEEE Proceedings of the Custom Integrated Circuits Conference.* New York: IEEE, 1984, pp. 117–121.
7. Batali, J. An Introduction to DPL. MIT Memo 81-65, October 1981.
8. Shrobe, H. E. "The Datapath Generator." *Proceedings of the Conference on Advanced Research in VLSI,* MIT, Cambridge, Massachusetts, 1982. Dedham, Mass.: Artech House, 1981, pp. 175–181.
9. Agre, P. E. A High-Level Silicon Compiler. Ph.D. Thesis, Massachusetts Institute of Technology, January 1983.
10. Gajski, D. D., and R. H. Kuhn. "New VLSI Tools," *IEEE Computer Magazine,* December 1983, pp. 11–14.
11. Chandramouli, R. "Designing VLSI Chips for Testability," *Electronics Test,* November 1982, pp. 50–60.
12. Davis, R., and H. Shrobe. "Representing Structure and Behavior of Digital Hardware," *IEEE Computer Magazine,* October 1983, pp. 75–82.
13. Kraft, A. "XCON: An Expert Configuration System at Digital Equipment Corporation." In P. H. Winston and K. A. Prendergast (eds.), *The AI Business.* Cambridge, Mass.: MIT Press, 1984.
14. Kim, J., and J. McDermott. "TALIB: An IC Layout Design Assistant," *AAAI Proceedings of the National Conference on Artificial Intelligence.* Los Altos, Calif.: William Kaufman, 1983, pp. 197–201.
15. Thomas, D. E., C. Y. Hitchcock III, T. J. Kowalski, V. J. Rajan, and R. Walker. "Automatic Data Path Synthesis," *IEEE Computer Magazine,* December 1983, pp. 59–70.
16. Forgy, C. L. *OPS5 User's Manual.* Carnegie-Mellon University Report CMU-CS-81-135, July 1981.
17. Southard, J. R. "MacPitts: An Approach to Silicon Compilation," *IEEE Computer Magazine,* December 1983, pp. 74–82.
18. Gajski, D. D., and J. J. Bozek. "ARSENIC: Methodology and Implementation." *IEEE Proceedings of the International Conference on Computer-Aided Design.* New York: IEEE, 1984, pp. 116–118.
19. Brown, H., C. Tong, and G. Foyster. "Palladio: An Exploratory Environment for Circuit Design." *IEEE Computer Magazine,* December 1983, pp. 41–56.

Panel: Artificial intelligence tools in actual use—I

Chair:
JAMES SLAGLE, *University of Minnesota,* Minneapolis
Members:
B. CHANDRASEKARAN, *Ohio State University,* Columbus, Ohio
NANCY MARTIN, *Wang Institute of Graduate Studies,* Tyngsboro, Massachusetts
JOHN VITTAL, *Xerox Corporation,* Pasadena, California

Expert systems can be built from scratch, using programming languages such as LISP, Prolog, or even FORTRAN; but recent increases in the understanding of the common patterns that appear in such systems have led to the creation of tools called *shells* for building expert systems. Such tools are higher-order languages, independent of particular application, that attempt to provide a user-friendly interface, a general-purpose inference mechanism, and a knowledge representation paradigm such as frames or rules. These shells can greatly increase the speed with which a new expert system is implemented. This session describes and contrasts some of the expert system building tools that have recently become available.

Panel: Artificial intelligence tools in actual use—II

Chair:
EAMON BARRETT, *Smart Systems Technology,* McLean, Virginia
Members:
RUBIN BROOKS, *ITT Research Laboratories,* Shelton, Connecticut
THOMAS BYLANDER, *Ohio State University,* Columbus, Ohio
JOHN HINCHMAN, *General Dynamics,* San Diego, California

Expert systems have excited the imagination of all those seeking increases in productivity from their computer systems. Although there is much activity in the field, a relatively small number of expert systems are in actual production use. This session focuses on existing, economically viable expert systems and some soon to be installed, addressing applications in computer system configuration, the petrochemical industry, and the financial and military arenas.

Panel: Qualitative reasoning for prediction and diagnosis

Chair:
KENNETH FORBUS, *University of Illinois—Urbana,* Urbana, Illinois
Members:
B. CHANDRASEKARAN, *Ohio State University,* Columbus, Ohio
JOHAN deKLEER, *Xerox PARC,* Palo Alto, California
JOHN MOHAMMED, *Schlumberger,* Palo Alto, California
RAMAN RAJAGOPALAN, *IBM,* Houston, Texas
REID SIMMONS, *Massachusetts Institute of Technology,* Cambridge, Massachusetts

Classical numerical techniques are insufficient to make computers that can analyze, monitor, operate, and repair complex physical systems as well as people do. The tacit expertise people bring to bear on these tasks, the common-sense knowledge about the physical world gleaned by years of living in it, must also be captured. Qualitative physics is the attempt to formalize this tacit knowledge and endow computers with similar reasoning skills. The members of this panel, representing several different approaches to qualitative physics, contrast their systems and explore potential applications.

Panel: Knowledge-based systems for engineering design

Chair:
DUVURRU SRIRAM, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
Members:
STEVEN J. FENVES, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
JIN H. KIM, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
L. STEINBERG, *Rutgers University,* New Brunswick, New Jersey

Knowledge-based expert systems are emerging as an important tool kit for the development of engineering software. These systems incorporate the heuristic knowledge of experts. Since a large part of engineering design is heuristic, these expert systems provide a means for automating the design process. Engineering design involves visualization of the product at the highest level; as the design progresses, this abstraction is refined into smaller subsystems. Since design involves subdividing the problem, interactions among subproblems must be carefully handled.

This session reviews the current state of the art of the application of knowledge-based expert systems to engineering design. The panelists develop a common methodology (problem-solving process and constraint handling) that arises from the approaches discussed.

Panel: Applied artificial intelligence: Future or fantasy?

Chair:
MICHELE K. PESTA, *AT&T Information Systems,* Summit, New Jersey
Members:
KEN BECK, *Texas Instruments,* Austin, Texas
C. KERRY NEMOVICHER, *Consultant,* Morganville, New Jersey
DENNIS O'CONNOR, *Digital Equipment Corporation,* Hudson, Massachusetts
DAN SCHUTZER, *Citibank,* New York, New York

In its July 9, 1984, cover story, *Business Week* proclaimed: "Artificial Intelligence—It's Here!" A scant few weeks later *Fortune* magazine, in its August 20 issue, reported that "Programs called expert systems are being ballyhooed as the hottest technology around. While useful for some tasks, the systems aren't as smart as they sound."

No longer the exclusive toy of academia and esoteric research institiutions, the buzzwords and catch-phrases of AI have found a prominent place in the popular media. Throughout the realm of applied technology AI is now the subject of an ongoing debate, its virtues both touted and doubted by a community anxious to see real progress. Skeptics remain unconvinced. For the casual but interested observer, most of the issues still remain clouded in jargon and partially understood concepts.

This session attempts to cut through that fog—to clarify terms, concepts, and issues. It does not attempt to resolve the debate of what is possible versus what is merely potential, but rather to set up a framework within which meaningful dialogue is possible.

Panel: Silicon compilers

Chair:
DANIEL GAJSKI, *University of Illinois—Urbana,* Urbana, Illinois
Members:
HAL ALLES, *Silicon Design Labs,* Liberty Corner, New Jersey
WALT CURTIS, *Silicon Compilers, Inc.,* San Jose, California
DOUG FAIRBAIRN, *VLSI Technology, Inc.,* San Jose, California
ROBERT KUHN, *Gould Research Center,* Rolling Meadows, Illinois
GARY MILES, *Seattle Silicon Technology, Inc.,* Bellevue, Washington

Silicon compilers represent a new technique for designing complex integrated circuits. By automating the block, logic, circuit, and layout stages of the design, the engineer is free to concentrate on the architectural specification. This session presents the state of the art in silicon compilation and describes some of the available systems.

Panel: Future of automated reasoning

Chair:
LAWRENCE WOS, *Argonne National Laboratory,* Argonne, Illinois
Members:
WOODY BLEDSOE, *MCC,* Austin, Texas
LARRY HENSCHEN, *Northwestern University,* Evanston, Illinois
DOUG SMITH, *Kestrel Institute,* Palo Alto, California

Previously open questions in mathematics and logic have been answered, superior logic circuits designed, the design of existing binary adders validated, claims about computer programs proved, computer programs synthesized—all with the assistance of various automated reasoning programs. How much did the program do, and how much did the person do? You do not need to be an expert in automated reasoning to use such a program. Are there differences between automated reasoning and artificial intelligence? Can a single reasoning program provide assistance in all the areas cited, and if so, what does that say about general-purpose versus special-purpose programs? In addition to the use of parallel processing, the panel will discuss what is needed to make reasoning programs more powerful and more useful. For example, how can reasoning programs provide greater assistance for systems control, diagnosis, and design? Finally, is the future of this new field as challenging, exciting, and promising as some imply?

FUTURE ARCHITECTURES AND SUPERCOMPUTERS

KAI HWANG, Track Chair
University of Southern California
Los Angeles, California

The architecture and implementation of the FLEX/32 MultiComputer

by NICHOLAS MATELAN
Flexible Computer Corporation
Dallas, Texas

ABSTRACT

The FLEX/32 MultiComputer is a generic environment for cooperating multiple processors. The FLEX/32 can support a number of different processors, making it heterogeneous in terms of the instruction sets it can support, and homogeneous in its ability to provide consistent storage and input/output facilities to its differing processors. These facilities are accessed through standard 32-bit VMEbus connections.

The FLEX/32 supports the full UNIX System V Operating System and languages associated with it, plus the extended ConCurrent C, ConCurrent FORTRAN 77, and Ada languages that allow programming of concurrent software at a high level. Direct programming support at all levels is provided by the environment hardware for concurrent software execution and optimization, including hardware support for shared resource access arbitration, conditional critical region arbitration, and interprocessor messages.

INTRODUCTION

The past 30 years have seen dramatic improvements in the performance of computers. In general, these improvements have been due to improvements in components. Today, as we approach physical limits in the performance of components, the architectures of computers become more and more important. Though we still see a great deal of improvement to be made in the application of faster component technologies, the effects of these components on conventional architectures using a single central processing unit begin to show diminishing returns. Architectures based on multiple central processing units show promise of providing not only increased power, but also increased flexibility in meeting the varying demands of future computation. The FLEX/32 is such a system (see Figure 1).

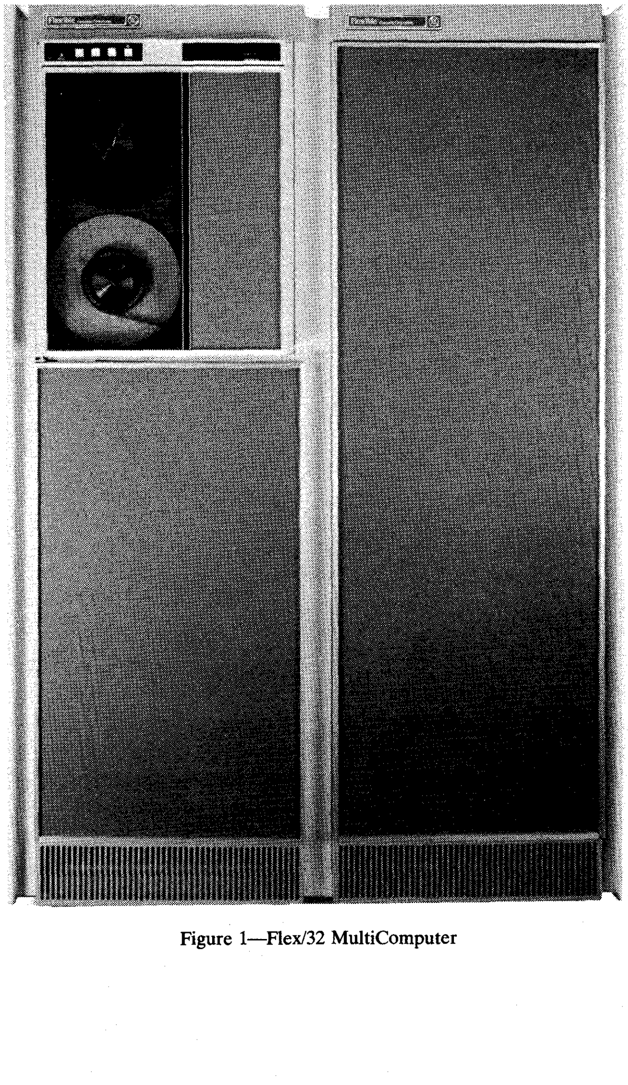


Figure 1—Flex/32 MultiComputer

The FLEX/32 is a MultiComputing Environment; that is, it is an environment that supports multiple computers working on one or more tasks together or independently under coordinated software. These computers need not be the same. Therefore, the environment is heterogeneous. The computers supported in this environment can differ in power, in the amount of memory supported, and in their basic orientations. Some of these computers could contain processors dedicated to control; others might be used for array processing and floating-point operations, for example.

The addition of a new computer, with its new processor and instruction set, requires only adaptation to the environment. This means that once the adaptation has been made, the generic software and input/output capabilities of the environment are fully available to the new processor. Such software includes the UNIX System V Operating System and other special tools needed for developing concurrent programs.

Input/output is performed via a set of standard VMEbuses. These buses support interfaces to peripheral equipment that may be purchased from any of the 80 to 100 current providers of VMEbus interfaces or from Flexible Computer Corporation, giving a truly open architecture.

A final, but no less significant, feature of the environment is its SelfTest System. Built into the environment and distributed throughout its modules (computers, memory, and peripheral interfaces) are test circuits dedicated to determining the health and performance of the environment as a whole. This system allows not only such features as automatic shutdown and restart in response to power failures, but also fault isolation and repair verification and performance analysis based on information collected during the run-time execution of programs.

LEVELS OF DESCRIPTION

In describing a computer, it is important to distinguish between its architecture and its implementation. An architecture is a description of the fundamental attributes and functionality of a device or program without regard to its detail. An implementation is a description of the collection of details needed to construct a product. It is the product that provides certain levels of performance and power. It is the architecture that provides product line consistency throughout its history.

Both hardware and software have an architecture. A major thrust of the FLEX/32, in addition to its great flexibility of configuration, is to provide an environment for programmers to produce software more cost effectively. The dichotomy of the architecture and implementation is maintained primarily to guarantee that software interfaces to the FLEX/32 MultiComputing Environment will not change throughout the life of the product line. We all know that as time passes there will be better ways to produce hardware that is faster, more reliable, and cheaper. However, it is the intention of the Flexible Computer Corporation that changes in FLEX/32 hardware, as they must come, will not involve changes in existing application software. The architecture defines a set of invariant interfaces for the users and builders of software intended for the FLEX/32 product line, even as its hardware is adapted over the years to the use of newer and better techniques. Its software, at every level, can remain invariant because its interface to the hardware remains invariant.

THE FLEX ARCHITECTURE

The FLEX/32 is a Multiple Instruction Stream/Multiple Data Stream (MIMD) Multiprocessor System. Its architecture (generically represented in Figure 2) is composed of devices, buses, processes, and topologies.

Devices are either computational or peripheral. Computational devices include processors, memories, bus interfaces, interprocessor signaling mechanisms, and common locks. Peripheral devices are the devices that control and sense the outside environment. They include tapes, disks, printers, terminals, and their controllers.

Devices are connected one to another by buses. Four separate buses are defined in the FLEX/32 architecture. These are the common bus, the local bus, the peripheral bus, and the SelfTest Bus. Common buses are those that are intended to allow the sharing of resources, such as shared memory or shared devices. Local buses are intended to provide paths between a processor and its attached local devices, such as local memory. Since each local bus is defined as multimaster, more than one processor can be attached to it.

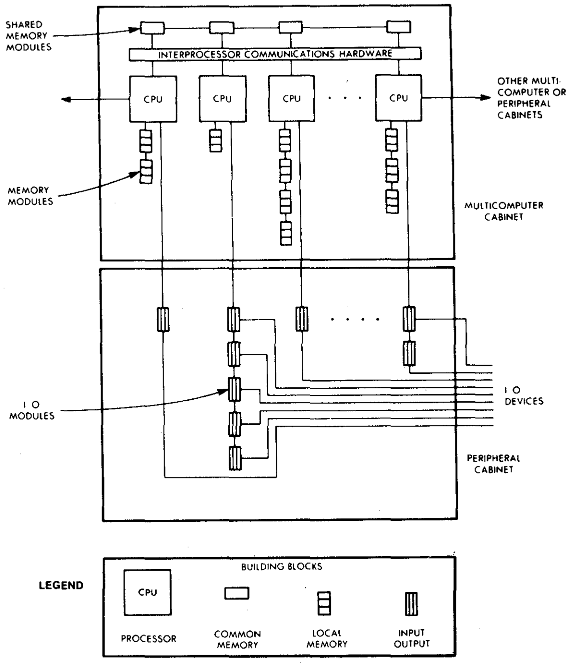


Figure 2—Generic flex architecture

Peripheral buses are those that allow attachment between devices on local buses to the controllers of peripheral devices and actuators in the outside environment. The SelfTest Bus, which runs throughout the MultiComputing Environment, is that transfer path that allows information about the state of the machine to be received and control of the machine to be sent.

Of course, all this hardware would be useless without a method for applying algorithms to control and direct operations toward a useful result. The FLEX/32 supports multiple, truly concurrent processes. True concurrency is the execution of processes on different processors at exactly the same time. This is often called multiprocessing. Apparent concurrency is the execution of processes in a shared fashion on a single processor at a rate fast enough to lead an observer to believe that they execute at the same time. This is usually called multitasking or multiprogramming. The FLEX/32 is a parallel processing system and therefore allows both true concurrency and apparent concurrency, depending on the needs of the programmer.

Each of the devices, processes, and buses outlined above may be connected in number of possible configurations. Figure 2 shows the physical topology of the FLEX/32 MultiComputing Environment. Any number of local buses may be attached through ports to several common buses. Common buses are fully arbitrated and will allow only one access at a time from each local bus in a fair arbitration scheme. Each local bus is attached to its own common lock device. This device can be programmed to allow processes mutually exclusive access to shared resources for an arbitrarily complex use of shared data. In general, interprocessor communication is maintained through shared memories residing on the common bus. Notice that this topology does not allow processors to reside on the common bus, only resources. Processors are always attached to local buses. Peripheral devices are attached to local buses through bus interfaces.

There is no limit to the number of local buses or the number of processors that may be attached to local buses in the architecture. Implementations, however, must put a bound on the number of buses and processors that can be contained in any particular unit, such as a cabinet. The architecture, however, fully allows multiple cabinets; therefore, even in the implementation there is no inherent bound on the number of processors that can be placed together in a FLEX/32 configuration.

The SelfTest Bus connects all devices in the architecture. Processes that actually perform selftest functions reside within one of the processors in the system.

THE FLEX/32 HARDWARE IMPLEMENTATION

The FLEX architecture is an abstract representation of the allowable interconnections of devices. As such, it makes no demands for execution speed, word sizes, and so forth. The FLEX/32 product line is a 32-bit implementation of the FLEX architecture. It specifies the technology from which the FLEX/32 is built, its packaging, and its physical parameters. The FLEX/32 hardware implementation is the environment used to carry out the commands of the FLEX software decribed below.

The philosophy of the FLEX/32 implementation is the provision of a flexible, universal hardware and software environment for a number of different instruction sets. Just as software environments have become a way of supporting programs written in different languages, the FLEX/32 hardware is an environment for different processors. These processors are supplied with generic memory, input/output support, and multicomputer and network interprocess communication mechanisms. This environment is the same for each processor type, but the instruction sets supported are different. This allows not only software written in different languages, but also different machine instruction sets, to execute together.

The components of the FLEX/32 hardware are divided into the Card Level, the Backplane Level, and the Unit Level. Circuit cards define the replaceable module level of the FLEX/32. There are three classes of cards in a FLEX/32 System. These are Universal Cards, Common Communication Cards, and Peripheral Cards.

Universal Cards support local bus activities such as computation, memory storage, and array processing. Common Communication Cards allow access to, arbitration of, and control over the common buses and their shared resources, such as the common memory. Peripheral Interface Cards are standard single, dual, or triple Eurocard interfaces available from commercial manufacturers. There are currently 80 to 100 commercial manufacturers producing cards to control standard peripheral equipment based on the VMEbus Eurocard format. These cards are interconnected via a backplane supporting the local bus, common bus, and SelfTest Buses.

The local bus is a standard, asynchronous VMEbus with extensions for control internal to the FLEX/32 cabinet. The common bus is a synchronous version of the local bus. The Common Communication Cards house a high-speed shared memory. The SelfTest Bus is an RS422 bus supporting the SDLC protocol. All external communications are through bus interfaces on Universal Cards to standard VMEbuses (no extensions). Peripheral buses are attached to standard VMEbus Interface Cards.

The cards connected to the backplanes are supported by card cages and are divided into two types of units. One unit is the MultiComputer Unit, or MU. It is the MU that houses both Universal Cards and Common Communication Cards. All Interface Cards to peripheral equipment are housed in a second type of unit, called a Peripheral Control Unit, or PCU. Cables between the two units allow computers attached to local buses to control their various I/O devices.

The MultiComputer Unit can house up to 10 local buses and 2 common buses. There can be 20 universal cards in a MultiComputer Unit, 2 per local bus. The MultiComputer Unit supports up to 10 communication cards, 1 for each local bus, allowing any processor attached to any local bus to communicate with any other processor, either through a shared memory associated with a Common Communication Card or through the direct interprocessor messaging capability. Furthermore, a common lock capability allows a processor to define a critical region in the shared memory and to own that region for operations without affecting traffic on the common bus.

The initial Universal Cards offered by Flexible Computer Corporation are Computer Cards and the Memory Card. Computer Cards, based on either a 32032 or a 68020 processor, include an attached floating point coprocessor and 128K bytes of ROM, one or four megabytes of memory protected by error correction and detection logic, and a VMEbus port that can be configured to either 32 or 16 bits of data. The processor can access its bus interfaces and its memory on this card without affecting the operation of its local bus. This is important when both slots associated with local buses are used for computer cards.

The Memory Card supports from 1 to 8 megabytes of random access memory protected by error correction and detection logic. It also contains a VMEbus interface.

The MultiComputer Unit can be configured in a number of ways. For example, a unit can be configured with 20 computer cards, giving a machine with 20 processors, 20 to 80 megabytes of memory, and 20 VMEbus interfaces. This system could also be configured to support up to 20 megabytes of fast common memory. Another system could be configured with one computer card, and the remainder of the MU filled with memory. This would give a system with 1 processor and 20 VMEbus interfaces plus 153 megabytes of memory. A more usual card complement would be four or five computer cards with an extra Memory Card, giving processors with 9 megabytes of memory each and perhaps a few single processing cards without extra memory. Other computer card types will include floating-point capability in the 4-to-6-megaflops (millions of floating-point operations per second) range. It should be noted that all FLEX/32 processor types can be mixed and matched in each FLEX/32 MultiComputer Unit.

VMEbus interfaces can be simply cabled together, giving extra shared paths other than those associated with the common buses. Interprocess communications over these paths can be made using read/modify/write interprocess communication instructions between local memories. The same jumpers can be used to connect multiple MultiComputer Units, forming much larger systems. Four of these VMEbuses, for example, could be used to connect to neighbors north and south, and east and west. Such a method could be used to define a plane of MultiComputer Units. Similarly, six interconnections could be used to define hypercubes of MultiComputer Units yielding a large number of computers (dozens to thousands) that could be applied to the same tasks. The possibility of such large multiple-processor systems makes the selection of the algorithms very important in determining the usefulness of any configuration. As is the case with any computer, infinitely expandable may not mean infinitely useful, except for a narrow range of algorithms. It is fortunate, however, that some of these algorithms are very useful indeed.

FLEX/32 SOFTWARE IMPLEMENTATION

For system development, Flexible Computer provides the full UNIX System V Operating System supported on each computer within the MultiComputing Environment.

Flexible Computer has also extended the C and FORTRAN languages to produce the new languages ConCurrent C and ConCurrent FORTRAN. These languages are standard C and FORTRAN with an extra set of statements that allow direct specification of concurrent programs for execution in the FLEX/32 environment.

The FLEX/32 can execute programs directly under ConCurrent C or FORTRAN program control instead of under UNIX. Flexible supplies a set of MultiComputing Multitasking Support Utilities to facilitate such dedicated operation.

The Ada language is also available.

UNIX System V

UNIX System V is a true industry standard for software development. It includes support software such as Source Code Control System (SCCS) and its associated editors and language processors, such as FORTRAN 77, which Flexible has extended with the ISA real-time extensions (S61.1), RATFOR, SNOBOL, and Assembly Language. It provides development and debugging tools and file management capabilities within the most portable operating system presently available. In addition, concurrent execution of processes can be simulated, using the shared-memory software capability of UNIX System V, or truly executed simultaneously.

The ConCurrent C and ConCurrent FORTRAN Programming Language

The C Programming Language has proved an excellent tool for programming in a sequential processing environment. The ConCurrent C Language[1] is designed to further increase the capabilities of the C language by facilitating direct concurrent and real-time processing for advanced parallel multiprocessor systems while maintaining the original C style and philosophy. C is upward compatible to ConCurrent C, which preserves all of C's definitions and features.

ConCurrent C extended constructs are categorized in two classes: new variable definitions (event variables and shared variables) and new control-flow statements (process interaction, process control, concurrent execution, and event supervision).

ConCurrent C introduces a new type, event variables, to support real-time event handling. All real-time events are either timers or exceptions. The keywords *timer* and *exception* are used to declare and define event variables.

The WHEN statement is used to suspend its enclosing process until a specified event is satisfied, at which time its associated statement list is executed. The WHENEVER statement is provided to support nonprocedural event supervision.

The WHEN statement is also used, in addition to procedural event response, to synchronize access to shared data between processes. The WHEN statement can thus be used to directly implement the Conditional Critical Region technique of sharing data.

The process concept is the basis of true concurrent execution. A process in ConCurrent C is defined and started by a PROCESS statement.

Proper combinations of these statements and other existing C Language statements can define and cause to be executed every known multiple process intercommunication technique, including semaphores, monitors, and messages.

Each of the capabilities of ConCurrent C listed here is also available in Flexible's ConCurrent FORTRAN 77.

MultiComputing Multitasking Operating System

The FLEX/32's MultiComputing Multitasking Operating System (MMOS) provides support for real-time, run-time embedded applications.

The MMOS Utilities are resident in the System Library and are included by the loader to resolve all external calls generated by the ConCurrent C preprocessor. The capabilities of the MMOS Utilities include the following:

1. Priority Oriented Task Management and Multiprogramming
2. MultiComputing, by providing interprocessor communication, synchronization, and data protection for concurrent or sequential processing
3. Interprocess Communication and Signaling
4. Event Management to supervise conventional interrupts, Interprocessor Messages, user-defined exceptions, system defined exceptions, and timers
5. Memory Pool Management

A METHODOLOGY FOR CONCURRENT PROGRAM DEVELOPMENT

Of major importance in producing commercial-quality (that is, useful) concurrent programs is the availability of a development methodology fully supported by software tools.

Figure 3 provides a block diagram of the flow of program compilation, loading, and execution within the FLEX/32 MultiComputing Environment. The steps from sequential code development through final concurrent program integration constitute the FLEX/32 development methodology.

At the top of Figure 3, a ConCurrent C program source file is shown. It is first processed via the ConCurrent C Preprocessor, resulting in the output of a C Language Source File containing unresolved MMS system calls. The Preprocessed ConCurrent C Source is next compiled under UNIX System V by the C compiler, resulting in an object code file. The object code file can then run through the system loader and have all system calls resolved by the MMOS Utilities contained in the System Library, resulting in an executable image file.

The right side of Figure 3 depicts the several execution options provided by the FLEX/32 MultiComputing Environment. Three different environments are provided with the FLEX/32 MultiComputer. The first allows execution under a Concurrent Executive and provides for a true parallel-computer, concurrent operating environment. The second is execution in a simulation of a concurrent environment under

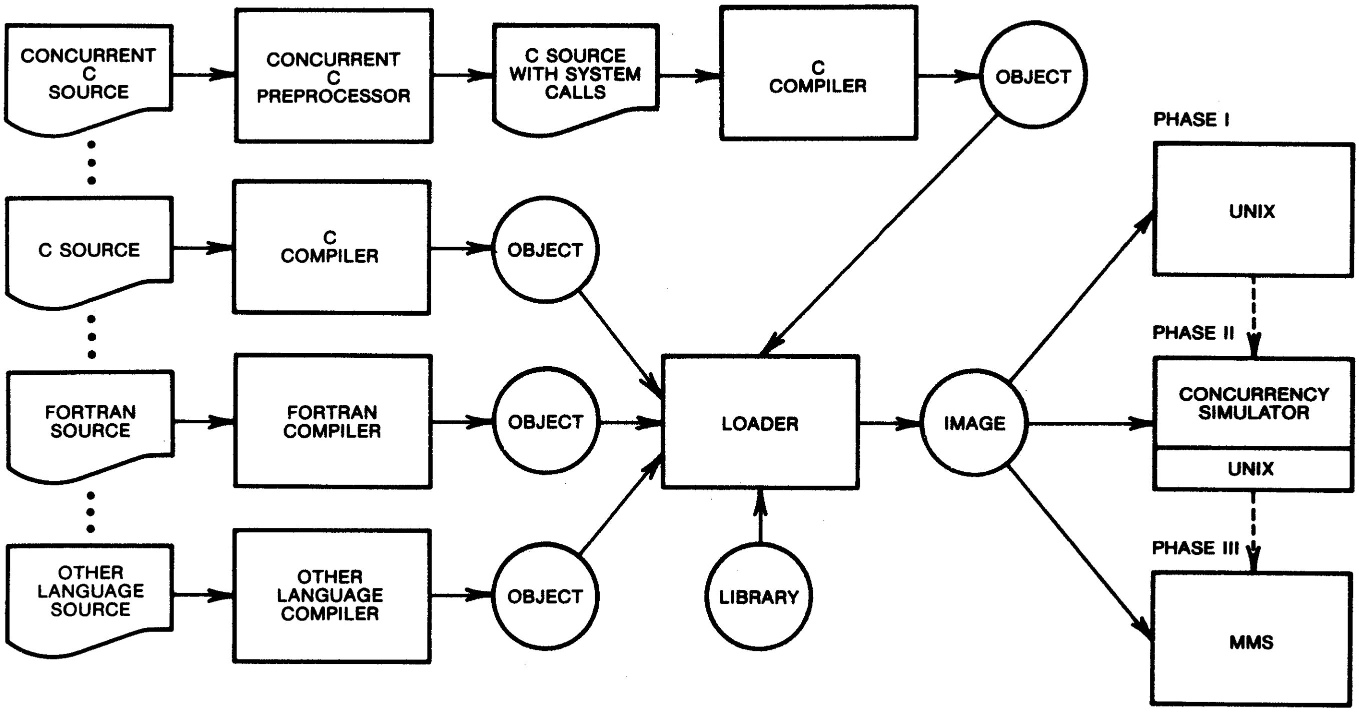


Figure 3—Software development methodology

the UNIX System V environment. This program is called the FLEX/32 Concurrency Simulator. The third environment is UNIX System V, which allows either nonconcurrent (sequential) operation or distributed operation.

The FLEX/32 methodology for concurrent program development is a phased migration of processes from one environment to another. Separate compilation and testing of sequential programs under UNIX and its tools allow a number of programmers to contribute to the development of large software systems. As these programs are developed and known to be functionally correct, they can be collected under a ConCurrent C program as processes. This concurrent program can be fully debugged, using the Concurrency Simulator, still under UNIX and its many support tools. If appropriate, this program can be forever executed in multiple processors under UNIX. If, however, the program was intended to execute directly on the MultiComputer, processes can be moved one at a time, or all at once, into the intended processors under MMOS. This allows incremental use of true concurrency from the shelter of apparent concurrency and UNIX support afforded by the Concurrency Simulator.

The diagram also depicts a number of source processes written in some of the languages that can be compiled via their individual language compilers and combined by the system loader. Again, their system calls are resolved, and images produced that allow them to execute together in different computers under any of the concurrent, simulated concurrent, and nonconcurrent environments provided.

CONCLUSION

Concurrent processing using multiple processors is an advanced technique for achieving more processing power, faster computation, and flexible application of computing hardware to changing requirements.

ConCurrent C and FORTRAN were developed to provide high-level software development tools for the concurrent programming of true multiple instruction stream/multiple data stream (MIMD) computing environments such as the FLEX/32 MultiComputer.

REFERENCES

1. Naeini, Ray. "The ConCurrent C Programming Language." *Electronics,* 57 (1984), pp. 125–129.

The Encore Continuum: a complete distributed work station–multiprocessor computing environment

by C. GORDON BELL, HENRY B. BURKHART III, and STEVE EMMERICH
Encore Computer Corporation
Wellesley Hills, Massachusetts

ANTHONY ANZELMO, RUSSELL MOORE, and DAVID SCHANIN
Hydra Computer
Natick, Massachusetts

ISAAC NASSI
Encore Languages
Wellesley Hills, Massachusetts

and CHARLÉ RUPP
Resolution Systems, Inc., an Encore Company
Marlboro, Massachusetts

ABSTRACT

The Encore Continuum is a UNIX-compatible computing environment designed to provide a range of computing styles from distributed work stations to host multis. The *multi,* meaning multiple microprocessor, is a new computer class that spans an order of magnitude performance range covering large micros to small mainframes. The multis' parallelism can be exploited for transparent timesharing, transaction processing, and real time control. Multis are also likely to be the basis for parallel processing.

INTRODUCTION

Companies have traditionally used multiple bit-slice technologies to produce computer "families" that cover a particular price/performance range. Powerful MOS and CMOS microprocessors are likely to change this strategy. Some companies have built proprietary microprocessors to implement new computers in their range (e.g., IBM's PC/370, DEC's MicroVAX and Data General's Micro-Eclipse). Others have built new computer families that leverage off of the performance range of standard microprocessor families (e.g., IBM's PC and AT).

A multiprocessing approach with up to four processors to achieve a performance range has been used by a few mainframe vendors over the last two decades, but the high interconnection costs between processor, memory and input/output components has limited the success of multiprocessing in range and applicability.

The Encore Continuum is a full-range computer family consisting of multis, distributed processing servers, high-resolution terminals, and work stations—all interconnected by local area networks. Microprocessors offer sufficient performance to support today's general-purpose computing at nearly all price/performance levels provided that the microprocessors are organized in groups. The Encore Continuum architecture is designed to exploit this rapidly improving technology; its multi scales linearly in price and performance over an order of magnitude range providing a high performance computation and database server, and its distributed work stations similarly scale over a factor of several hundred and provide both local processing and access to shared multis.

GOALS OF THE ARCHITECTURE

Major objectives of the hardware architecture include

1. Cost-effective multi-user computing with incremental scalability of an order of magnitude range for processor, memory, mass storage, and input/output computing resources.
2. Hardware-independence of microprocessor architecture, data formats (e.g., floating point), and communications technologies in order to track transitions to new technologies without rendering the architecture or user's system obsolete.
3. Arbitrarily large virtual memories to support memory-intensive applications, and the accommodation of very large physical memories for high-performance memory applications and input/output (i.e., file) buffering.
4. Ability to expand or rearrange systems easily to respond to changing requirements, new applications, and new computer and communications technology.
5. Access to the Continuum both within a local area and from long distances using industry-standard local and wide area network protocols from a variety of industry-standard user-interface devices including terminals and personal computers (PCs).
6. High-quality human interface devices with full-page display formats, integral graphics, distributed windowing, and industry-standard connections, protocols, and application software environments.

The goal of the Encore Continuum is to provide a complete and industry-standard compatible software environment that

1. Is UNIX-compatible for multiprogramming in order to achieve long-lived and stable system interfaces and provide for the acquisition of a large flow of compatible and competitive software from many sources. By having a single standard interface, applications software can be written to run on all hardware. Thus, the software industry competes to provide better products.
2. Is location independent so that a user can compute or store data with minimal communication costs, system responsiveness or performance is enhanced, and data and equipment security is increased.
3. Supports multiprogramming, distributed computing, and parallel processing applications with minimal degradation of throughput and response-time due to systems software overhead.
4. Provides identical programmer and user interfaces across the Continuum in order to protect investments in applications programs and personnel training, and enforce security restrictions uniformly, where needed.
5. Allows sharing of resources among nodes in the Continuum and other vendor equipment.

ENCORE PRODUCT OVERVIEW

Figure 1 shows the hardware products of the Encore Continuum as described below:

Multimax: A multiprocessor that spans a processing performance range of 1.5 to 15 million instructions per second (Mips) in increments of 1.5 Mips. Input/output throughput can be expanded in increments of 1.5 megabytes per second to 15 megabytes per second. Memory can be expanded in increments of 4 megabytes to 32 megabytes.

Annex (Ancillary Network Exchange computers): Intelligent, low-cost terminal and PC concentrator and gateway

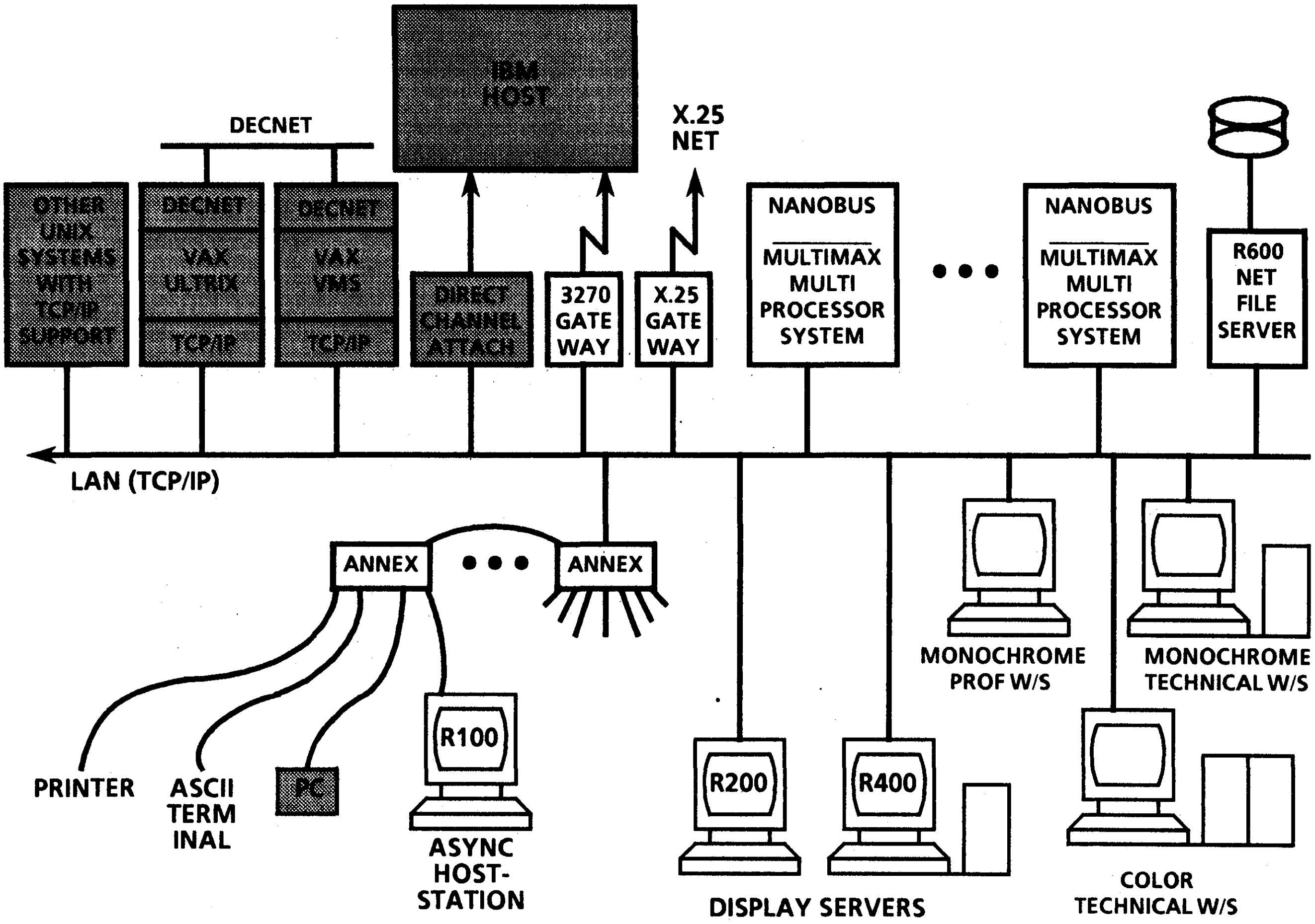


Figure 1—The Encore Continuum, a distributed processing and multiprocessor computing environment

computers for Multimax systems and Resolution stations. Functions such as terminal access to the Continuum and gatewaying to PCs, public data networks, and external computing environments such as SNA take place via Encore's Annex computers.

Resolution R100 Host stations and R500 Work stations: The R100 is a high resolution, large screen, multiple-window, multiple-host access station designed for host-based computer access. The R500 is a compatible work station with local processing, primary memory, and mass storage.

Local Area Network (LAN): Ethernet (IEEE 802.3) LAN is used to interconnect all computing elements and provide computer-to-computer intercommunication, common gateways to other computers and public communications networks, and common access to terminals and PCs.

Interconnections with other computers which support the TCP/IP protocols.

UMax 4.2 and UMax V: Two UNIX software environments derived from the University of California at Berkeley BSD 4.2 and AT&T UNIX are provided. These are primarily for technical and commercial applications, respectively. Software includes the 200 general-purpose tools provided with UNIX, a full complement of user productivity tools, editors, languages and debuggers, and a relational database with Ally, Encore's 4th-generation language.

THE MULTIMAX

Encore's Multimax computer uses multiple microprocessors sharing a common memory to achieve a scalable large performance range, lower price/performance ratio over range, and increased reliability and availability over uniprocessor systems.

Multimax's power is derived from the Nanobus which interconnects 20 Multimax cards within a backplane (see Figure 2). Every 80 nanoseconds, a 32-bit address (giving a system-wide address of up to 4 billion bytes) and 64-bits of data can be transmitted. Thus, Nanobus has a data carrying capacity of 100 million 8-bit bytes per second. (By comparison, standard and emerging buses for multis are usually one-tenth to one-fourth as fast.) All cards can operate online, offline, or on a completely standalone basis for complete self-diagnosis.

The Nanobus provides the key to product longevity, as it is able to accept new higher-speed processors that will evolve with CMOS VLSI. Real-time data can be processed at up to

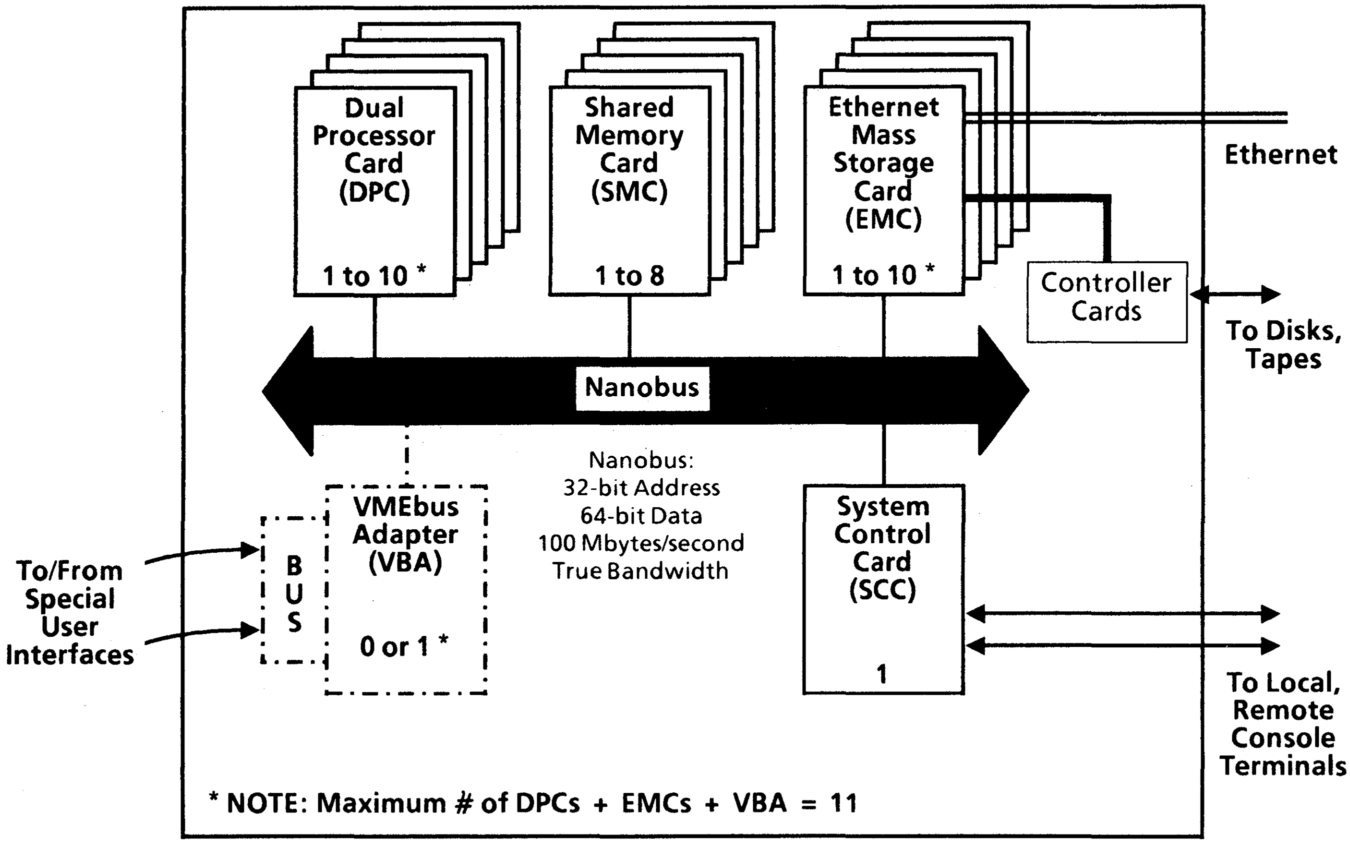

Dual Processor Card (DPC)
1 to 10 *
Shared Memory Card (SMC)
1 to 8
Ethernet Mass Storage Card (EMC)
1 to 10 *
Ethernet
Controller Cards
To Disks, Tapes
Nanobus
Nanobus: 32-bit Address 64-bit Data 100 Mbytes/second True Bandwidth
VMEbus Adapter (VBA)
0 or 1 *
BUS
To/From Special User Interfaces
System Control Card (SCC)
1
To Local, Remote Console Terminals
* NOTE: Maximum # of DPCs + EMCs + VBA = 11

Figure 2—The Encore Multimax multi(ple) microprocessor computer

full bus bandwidth (100 megabytes per second) using direct memory access (DMA) via priority-based programmed interrupts (40,000 events per processor/second maximum) and via direct program control.

At least one of each of the following card-type options is required:

1. Dual Processor Card (DPC)—two National 32032 processors share a common 32-kilobyte cache. A high-performance floating-point option using special chips is provided for arithmetic-intensive applications. Encore rates this processor at 0.75 million instructions per second. With 10 DPCs, a single Multimax can process up to 15 million instructions per second.
2. Ethernet/Mass Storage Card (EMC)—interfaces to Ethernet and to disks. This card contains a 32032 for managing input/output transfers and diagnostics. It has sufficient capability to operate as a LAN-based file service computer. Up to 11 DPC or EMC cards can be placed in one system.
3. Shared Memory Card (SMC)—four megabytes of memory organized in two independent banks with error detection and correction codes, an SMC uses 256-Kbit memory chips. Eight SMCs can be placed in a single system providing up to 32 megabytes of memory. An on-card computer can completely check and diagnose the memory offline.
4. System Control Card (SCC)—performs bus arbitration, logs errors, provides system diagnosis, and communicates with an operator and a remote console.

Peripheral Options

Multimax also offers battery backup—fixed and removable disks of 520 and 300 megabytes, respectively, and 1600/6250 bpi magnetic tape options.

THE LOCAL AREA NETWORK

Encore currently uses the most widely accepted standard (IEEE Standard 802.3—Ethernet) to interconnect its computing nodes at a rate of 10 million bits per second. Other standard LANs will be adopted in response to market requirements. The function of the LAN is for

1. Computer-to-computer communication for distributed processing, file transmission, and virtual terminal access among computers
2. Common access to other networks via Annex gateway computers
3. Common access from terminals and PCs via Annex concentrator computers

4. Formation of a fully distributed computing environment using Encore's powerful single-user work stations
5. Connection to existing personal computers, minicomputers, and mainframes

A LAN is not required for basic system operation.

ANNEX TERMINAL AND ACCESS (CONCENTRATOR) COMPUTER

Each Annex concentrator computer attaches up to 16 terminals and printers along the LAN in a fully distributed fashion permitting up to several thousand terminals to access all computers within a single LAN. Five Annex computers can be connected to a single LAN port, or it can be directly connected to a Multimax if there is no LAN. Annex roughly doubles the processing power of the Continuum, since roughly one Annex computer is used with each Multimax processor. Wiring from terminals to computer is simplified by distributing the physical connections to the Annex concentrators, unlike most terminal architectures, which require all RS-232C terminal lines to be connected to a particular computer. Since the LAN is basically a distributed switch, any serial port on an Annex can communicate with any Multimax or Resolution on the LAN. Annex incorporates a general-purpose remote procedure call facility to communicate with Multimax systems and Resolution work stations.

Annex is programmed to perform time-consuming functions such as character processing on input from, and screen updating on output to, terminals that require no host or central database interaction. Annex contains a National 32016 microprocessor with 128 kilobytes of memory.

On terminal initialization, a switch program asks the user to request a host. The Annex "notifies" the selected host, the terminal becomes connected to it, and the host runs the standard log-on process. Connection switching allows users to connect to other hosts and then to switch among them. Connection binding allows site managers to fix (bind) a port on a given Annex to a given host; this facility binds users to a particular application, and binds dedicated peripherals such as line-printers to particular computers.

Annex has options for both asynchronous and synchronous communications and direct and modem connections. Standard terminals and Encore host stations communicate with hosts at up to 38.4 kilobaud transmission rate per terminal. Hard copy options include a 200 character per second matrix printer and 800, 1200 and 1800 lines-per-minute printers.

ANNEX GATEWAY COMPUTER

The Annex gateway provides access to various communications and industry networks using protocol conversion hardware and software. The protocols include: IBM SNA, IBM Block Mode Terminals (3270), IBM PC and X.25. Presentation-level services associated with gatewaying generally run in the host, which communicates via the remote-procedure call communications architecture with protocol-conversion software running in the gateway.

RESOLUTION COMPUTING STATIONS

The Resolution stations use a 19-inch screen size to give an unscaled, ledger-sized 11″ × 14″ page at high resolution using 1056 × 864 pixels. A ledger sheet of 176 columns and 86 rows can be displayed. Keyboard and pointing device (e.g., mouse) input are provided. The stations (without keyboard) occupy a desk space of 16.5 square inches. The text and graphics protocols provided, which allow existing and future software to be run without modification, include VT100, ANSI 3.64, Tektronix 4010/4014, Regis, and VDI for GKS.

The stations are designed to address a variety of applications, including the station of choice for the professional programmer, text and typographic input, engineering, business and accounting where computational power and large screens are required, and special functions such as translation where side-by-side text is required.

Resolution Host Station (R100)

The R100 is a single, but universal host station because it can communicate with as many as three computers through separate windows. For example, the R100 can simultaneously access Hydra, a traditional host (e.g., IBM 370 or VAX-11), and a PC AT for personal computer software. All functions of the R100 are carried out under the program control of a National 32016 microprocessor. The R100 is also designed to be used as a remote slave station to conventional work stations (i.e., a user can have a work station at home or at a second office).

The R100 can be upgraded to become an R500.

Resolution Work Station (R500)

The R500 is a self-contained computer system with a primary memory of two megabytes and disk memory of 20 megabytes. The processor, a National 32016, is completely compatible with other computers in the Encore Continuum. Thus, software can be run either within the work station, among work stations, or among host stations and Multimax systems in a completely flexible and transparent fashion.

THE UMAX 4.2 AND V DISTRIBUTED SOFTWARE ENVIRONMENT

The software environment in the Encore Continuum is UNIX-compatible and enhanced to support both distributed and parallel processing. Distributed processing support is provided by a communications architecture that provides for cooperative, efficient interprocessing between networked Multimax systems, Resolution stations and Annex computers. Language constructs for assigning task forces of processors to a single process for support of parallel processing are also provided.

UMax 4.2 and UMax V constitute Encore's standard operating systems. Programs that run under either UNIX System V or UNIX BSD 4.2 are compatible and portable to the corresponding Multimax and Resolution systems.

In addition to UNIX standards, the Encore Continuum extends UNIX

1. To take full advantage of demand-paged virtual memory, multiprocessor performance, and distributed terminal architecture.
2. To provide data sharing and synchronization mechanisms between user processes (UMax 4.2). Additional system calls and library subroutines support these new multiprocessor functions.
3. By unifying language standards and language-related data formats across both operating systems, to simplify portability of applications between environments.

UMax Performance on the Multimax

UMax 4.2 and UMax V incorporate three strategies for high performance that are inherent in the Encore Continuum: (1) symmetrical multiprocessing, (2) scalability to a large number of processors, and (3) distributed intelligent peripheral control.

Symmetrical multiprocessing (multithreading) achieves maximal performance in the Multimax by ensuring that any processor can execute any user process or part of the operating system. This ensures that there are no inherent bottlenecks. One copy of the operating system supports all the processors, memory, and input/output computers. In order to allow multiple processors to gain simultaneous access to operating system services, concurrent access must be controlled to each process and operating system routine.

Controlled concurrent access to internal UMax resources is achieved with the following mechanisms:

1. Spin locks—accomplish synchronization by executing tight instruction loops until the expected condition occurs (used only for critical short-duration events).
2. Semaphores (Dijkstra style)—accomplish synchronization by putting requesting processes "to sleep" until the requested resource is available.
3. Read/write locks—specialized forms of semaphores; provide access to data structures for a single writer or multiple readers.

Scaling performance to many processors and very large memories is a major performance consideration. Multithreaded operation alone will not realize the performance potential inherent in the multiple resources of a Multimax. Two additional performance enhancements have been added to accommodate a large processing load: (1) shared data in the operating system is minimized, and (2) the UNIX terminal driver has been redistributed to the Annex concentrator computers. The first method caches frequently used in-memory resources such as file and directory entries. For resources controlled by tables, it is generally appropriate to lock individual entries rather than the whole table. In other cases, kernel tables have been divided into subpools of entries, linked together, and located by hashing. This minimizes search times and allows for locking of subpools rather than whole tables.

High-Level Languages and Debuggers

C. This language is supported by an optimizing compiler. Traditional assembly languages for system-level and time-critical applications programs are minimized.

FORTRAN-77. Fully conformant to the ANSI standard using an optimizing compiler.

Pascal. ISO standard.

COBOL 74. FIPS intermediate-level 2.

A source-level debugger for local debugging of user mode code, written in the C and FORTRAN-77 languages, is available. A remote debugger of supervisor mode programs that facilitates remote Encore diagnosis of system problems by the Encore service organization is also provided.

CONCLUSIONS

The indefinite expandability goals of the architecture are satisfied by allowing almost unlimited numbers of each system to be added to a local area network. Multimax and Resolution are incrementally upgradeable to higher levels of performance by adding processors, memory, and mass storage over an order of magnitude range. Standardization, portability, and ease of use are inherent with UNIX.

APPENDIX A
THE MULTI AS A NEW COMPUTER CLASS

The multi is an emerging computer class made possible by recently developed powerful micros that have the speed and functionality of mid-range super minicomputers. In contrast to computer families implemented from a range of technologies, a multi is scalable, thereby permitting the building of a single computer which spans a performance range. The multi is a significant alternative to conventional micros, minis, and mainframe families.

Multis can be used today, without redesigning or reprogramming of applications, because computer systems operate on many independent processes. With multis, it is possible to operate on many of these processes in a parallel fashion that is transparent to the user (with each process on an independent processor). Thus, the multi is likely to be the path to a fifth generation of computers based on parallel processing.

Historical and Technological Basis

Computer systems with multiple processors have existed since the second generation (the Burroughs B5000, a dual symmetrical processor, was introduced in 1961). Most mainframe vendors and some minicomputer suppliers currently offer systems with two to four processors. However, these structures have been expensive to build, due to the high cost of typical processors. Hence, they have found application mostly for high-availability computing (e.g., communications, banking, airline reservations).

The modern 32-bit microprocessor's function, performance, size, and relatively negligible cost create a new potential for multiprocessors. With 32-bit addressing, hardware support for page organized, virtual memory, and complete instruction sets with integer, floating, decimal, and character operations, these chips offer performance levels comparable to mid-range superminis such as the VAX-11/750.

The multi is a multiprocessor structure designed to take advantage of these new micros. It employs an extended UNIBUS-type interconnect whereby all arithmetic and input/output processor modules can access common memory modules. Cache memories attached to each processor handle approximately 95% of its requests, thereby limiting traffic on the common bus. With these local caches, more processors can be attached before saturating the common bus.

With proper attention to the design of critical elements (e.g., the common bus), large multis using current-technology micros can outperform high-end superminis and some mainframes in terms of total performance. This advantage should continue to grow. The performance of MOS and CMOS microprocessors has improved (and is expected to continue to improve) at a 40% per year rate while the TTL and ECL bipolar technologies (on which most traditional minis are based) have shown roughly a 15% per annum improvement.

When compared to traditional uniprocessor designs, the multi delivers improved performance, price, and price/performance as follows:

Configurability Range. Through modular design, the multi allows the user to "construct" the correct level of performance or price without having to choose among a limited number of computer family members.

Availability. The multi has inherent reliability through redundancy because it is built from a small number of module types (four, typically). With appropriate software support, faulty modules which are replicated can be taken out of service, thereby allowing continued operation with minimum downtime.

Design and Manufacture. Because the multi is composed of multiple copies of a small number of modules rather than the large number of unique boards in a typical minicomputer, it is faster and less expensive to design. Individual module types, produced in larger volumes, result in improvements of 30% in manufacturing costs over conventional uniprocessors.

Evolutionary Technology Upscaling. With appropriate design, multis allow long-term performance upscaling through evolution. As key components of the processor and memory cards improve over time, the computer can be upgraded without replacement, in an evolutionary fashion. In addition, increased cache sizes through denser parts and improved cache management disciplines will permit substantially greater numbers of processors to be installed without saturating the common bus (provided that the bus design has allowed for this performance growth). All of this will permit graceful and cost-effective evolution in processor performance, input/output throughout, and memory size over a range of one to two orders of magnitude over a ten-year period.

Applications

Multis will be widely used for many applications because they can provide the most cost-effective computation except in cases where the power of a single large processor is required on a single sequential program. Because of the rapid rate of microprocessor evolution, the percentage of applications requiring single-stream performance in excess of that delivered by each of the multi's processors is already small and will continue to shrink. On the other hand, the emergence of the multi will lead to parallel processing.

We can better understand how multis may be applied by classifying the degrees of parallelism achievable. *Grain size* (see Table I) is the period between synchronization events for multiple processors or processing elements. Synchronization is necessary in parallel processing to initialize a task, parcel out work, and merge results. The multi exploits coarse- and medium-grain parallelism within an application (not the fine-grain, which is the focus of vector pipelined computers such as Cray 1 on wide word microprogrammed array processors).

Table I—Constructs for parallelism, synchronization, and supporting Encore structures across various grain sizes

Grain Size	Construct for Parallelism	Synchronization Interval (Instructions)	Encore Computer Structures to Support Grain
Fine	Parallelism inherent in single instruction or data stream	1	Specialized processors (e.g., *systolic* or *array*) added to Multimax
Medium	Parallel processing or multitasking within a single process	20–200	Multimax
Coarse	Multiprocessing of concurrent processes in a multi-programming environment	200–2000	Multimax
Very Coarse	Distributed processing across network nodes to form single computing environment	2000–1M	Multiple Multimaxes, Encore work stations, and other machines, on Ethernet

Groups of multis and conventional work stations can interact over networks to implement very coarse granularity.

As all modern operating systems are multiprogrammed, whereby each job in the system is at least a single process, and many support multitasking or subprocesses, most current applications are already designed to take advantage of the multi at the coarse-grain level. When used in a timesharing or batch environment, each processor of a multi can be assigned to a separate job to exploit the parallelism inherent in the work load. The UNIX pipe mechanism allows multiple processes to be used concurrently on behalf of a single user or job to achieve parallelism in reading a file, computing, and outputting to one or more files. Transaction processing is inherently a pipeline of independent processes.

The multi can be a more efficient multiprogramming computer than the traditional uniprocessor because the number of context switches (and lost time) is reduced. Additional parallelism is in the operating system itself. Execution of operating system code often accounts for 25% or more of available processing time when file, database, and communications subsystems are included. By restructuring the operating system, multiple independent system functions can be executed on independent processors.

If reprogramming of subsections of the application is possible, multis permit additional parallelism to be realized at the medium-grain level (i.e., parallel processing) by segmenting a problem's data for parallel manipulation by independent processors. This has been shown to be quite effective on simulation, scientific modeling, and analysis problems (such as matrix operations, linear programming, and partial differential equation solution, etc.) that permit data elements to be processed in segments.

Finer granularity of parallelism is achievable in the framework of the multi through specialized processors installed into its common bus. This is most effective when the algorithms are known a priori as in certain signal processing applications.

Multiprocessors, augmented by both programmable pipeline (i.e., systolic) and specialized processors for fine-grain parallelism, will cover the widest range of problems of any computing structure.

Concurrent processing: A new direction in scientific computing

by JUSTIN RATTNER
Intel Corporation
Beaverton, Oregon

ABSTRACT

Although supercomputer-class performance is a necessity for many research applications, the supercomputer's high price tag places it far beyond the reach of most academic and industrial organizations. In addition, traditional shared-memory architectures, whether single-processor or multiprocessor, are rapidly approaching their performance limits. This paper describes Intel's new iPSC family—multiple-instruction, multiple-data (MIMD) machines implemented by a loosely coupled, distributed-memory, concurrent-processing architecture with a hypercube interconnect topology. The VLSI-based implementation of a concurrent architecture allows these systems to lower supercomputer costs significantly while offering unlimited increases in supercomputer performance. Such a system can provide a focus for parallel-processing research and lead to a greater availability of algorithms and software for a variety of applications.

INTRODUCTION: WHY A NEW DIRECTION?

In considering the way scientific computing is done today, two unmet needs become apparent. One is the requirement for a supercomputer that is more affordable by individual academic and industrial research groups. The other is the need for new techniques to take supercomputers beyond the performance limits imposed by today's single-processing architectures.

The Need for Lower-Cost Supercomputing

Supercomputers are an essential tool for research, design, and development. Yet despite their recognized importance, these machines are absent from many university and industry research facilities, where research is hindered by the lack of supercomputing power.

The primary reason for the short supply of supercomputers, outside a few government laboratories, is their cost: Most of the supercomputers available today (from manufacturers such as Cray Research and Control Data Corporation) carry a price tag in the $5 million to $15 million range. Universities and even many commercial users find such prices prohibitively expensive, even though these users recognize that supercomputers are often the most appropriate resource for performing certain complex and important tasks.

In this situation, scientists and others needing supercomputer power have turned to other kinds of computers to help them in their research. One alternative has been the superminicomputer, such as Digital Equipment Corporation's VAX. These systems have proved very popular because they are more affordable than supercomputers. However, their performance is limited—considerably less than 1 million floating-point operations per second (MFLOPS)—and as a result they cannot solve problems quickly. Some problems are actually beyond their capabilities. Thus, researchers frequently find themselves on the horns of a dilemma: They can scale back their problems to the computer's capabilities, in effect solving only part of the problems; or they can solve the entire problems, but at the cost of too much time, frustration, and monopolizing the machine for long periods.

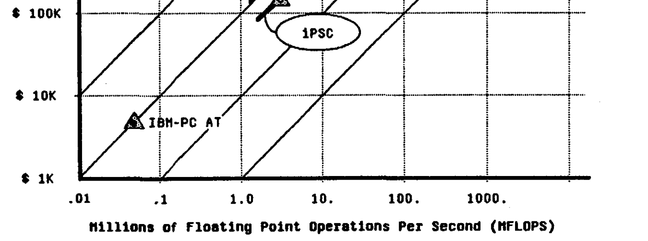
Another alternative for scientific computing has been the array processor, which attaches to a supermini or mainframe and typically delivers performance ranging from 1 to 10 MFLOPS. Array processors have less flexibility than a supermini or mainframe computer, however; and their cost, when added to the amount needed to purchase a host system, puts them out of reach for many researchers.

What is needed is a supercomputer that is affordable and accessible.[1] Currently, researchers need computers with performance of 1 to 100 MFLOPS, but at a cost low enough to justify dedicating the computer to a single project (Figure 1). Such a computer would eliminate the complications and cost of a large shared supercomputer, and it would permit individual researchers or small research groups to own and control the supercomputing resources they need.

The Need for A New Architecture

Soon the high costs of supercomputers will not be the only factor limiting their use. Conventional supercomputer architecture shows clear signs of reaching its theoretical performance limits within the next 5 to 10 years.[2] Already even the large Cray-class supercomputers, with peak performance in the range of 250 to 1,000 MFLOPS, are still inadequate for some types of problems. The demand for performance in some areas today is typically 10 times, and more often 100 times, the projected limit of 3,000 peak MFLOPS for current supercomputer technology.

In handling large-scale scientific and engineering problems that require a large number of calculations, existing supercomputers are essentially vector processors that depend on these data being in the form of either vectors or arrays. Though most scientific and engineering computations are dominated by vectors and arrays, and thus benefit from the



The current research need is for an affordable scientific computer with performance in the range of 1 to 100 MFLOPS. Because of cost, most researchers must force tasks onto affordable machines with insufficient capabilities.

Figure 1—The scientific computational environment (price vs. performance range)

vector performance of the supercomputer, a portion of any code will consist of scalar (single-quantity) operations that supercomputers must also be able to perform. For balanced performance, then, supercomputer architectures must accommodate both fast scalar processing and fast vector processing. A supercomputer handles scalar functions about as efficiently as a typical mainframe.[3]

How effectively a supercomputer handles the combination of vector and scalar operations is the key to its performance. Scalar operations are usually the limiting factor. For example, the peak performance rating of a supercomputer represents the maximum rate at which it can handle a completely vectorized problem. Yet the real performance does not meet peak ratings, because few, if any, problems can be fully vectorized. The parts of a problem that cannot be vectorized must run on the scalar portion of the system, resulting in an increase in overall computation time.

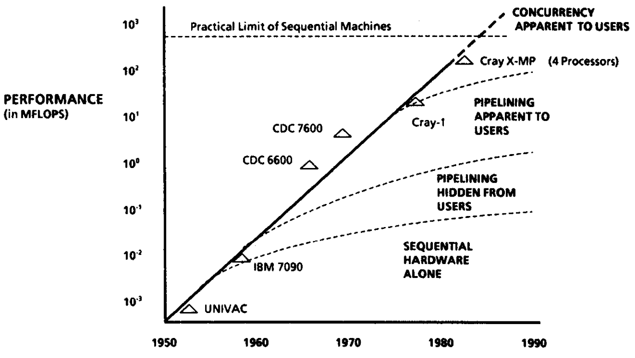
Pipelining is one of the primary methods used in supercomputers to speed performance, particularly vector performance. With the pipelining used in conventional supercomputers, arithmetic operations can be broken down and performed in assembly-line fashion, yielding faster performance. Such pipelining, however, makes programming difficult: Either the programmer must manage the time dependencies in the code, or sophisticated compilers must be used that recognize parts of the code that can be vectorized and handle the pipelining automatically. Ultimately, some hand-tweaking of the code is required to optimize performance.

In spite of the performance increases achieved through pipelining and vectorization, supercomputers are reaching the limits of their capabilities. This occurs because regardless of how sophisticated its design and how fast its components, a single-processor supercomputer eventually reaches limits imposed by fundamental electrical properties—switching speeds and propagation delays. The answer to improving supercomputer performance lies in concurrent architectures. Figure 2 shows the evolution of computer architecture and the performance limits characteristic of each stage in that evolution.

In sum, users with computationally intensive applications need machines with enough potential to meet the projected demand for ever increasing performance. With conventional supercomputer architectures close to their maximum performance levels, architectural breakthroughs are needed to meet the computing needs of these researchers well into the future—at a cost that more research institutions can afford.

CONCURRENT PROCESSING: AN APPROACH THAT MEETS BOTH NEEDS

Concurrent processing represents the most promising long-term approach to achieving affordable, accessible supercomputing.[4] Concurrency is a high-level or global form of parallelism, denoting independent operation of a collection of simultaneous computing activities. A concurrent machine thus uses loosely coupled, multiple, interacting processors to perform many operations at once. Concurrency contrasts with other forms of parallelism, such as pipelining and multiple functional elements. These forms imply some form of lock-step control, which ultimately limits the expandability and performance of a system. Concurrency allows expansion to a larger number of processors because of the flexibility afforded by distributed memory, distributed control, and loose coupling.



Courtesy: Bengt Fornberg, Exxon Research and Engineering

The continued advance of high-performance computing has been fueled by architectural innovation. Large-scale parallelism, the next major step, promises both lower cost and performance beyond the projected limits of today's supercomputers.

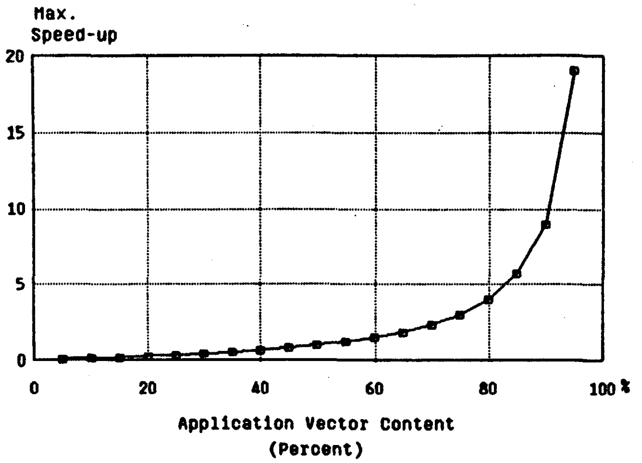
Figure 2—Trends in computer performance

To be solved on a concurrent machine, a problem is broken down into a number of pieces that can run concurrently on more than one processor. Fortunately, this fundamental structure is characteristic of a wide variety of applications, particularly in scientific computing.

Concurrent machines are the next logical step in higher-performance computing because they are able to effectively exploit the parallelism inherent in the physical structure of computational problems. As Charles Seitz suggests, "Concurrency is a fundamental aspect of nature" (personal communication, July 1984). When a computer is used to model a natural phenomenon, as in computational physics or chemistry, concurrency is the underlying operational principle. It is logical, then, that our computational tool should have the same properties that characterize the problem. Simply stated, concurrency makes use of the parallelism inherent in the problem—that is, multiple processes occurring simultaneously. A concurrent machine can, in a sense, become an electrical model of a physical system being studied.

The program to solve a complex mathematical problem is generally separated, conceptually, into a scalar portion (the outer loop or problem space), which mathematically describes the boundary of the problem; and a vector portion or inner loop. On a vector or array processor, the parallelism inherent in the vector portion of a mathematical problem also facilitates high performance.

However, the scalar portion of the problem limits performance, because it cannot be vectorized and must run sequentially. As Figure 3 indicates, infinite vector performance will have little effect on increasing the overall performance of the application beyond this point. It has become "scalar bound." This observation is supported by users who report that the



The vector/scalar ratio of a machine determines the class of problems for which it can be used efficiently. Given a normalized scalar performance of 1.0, the speedup observed by the addition of an infinite-performance vector processor is determined by the vector content of the computational problem. A 75% vector content yields a performance improvement of only 3×, while a 20× increase requires greater than 95% vector content.

Figure 3—Performance speedup vs. application vector content (infinite vector performance)

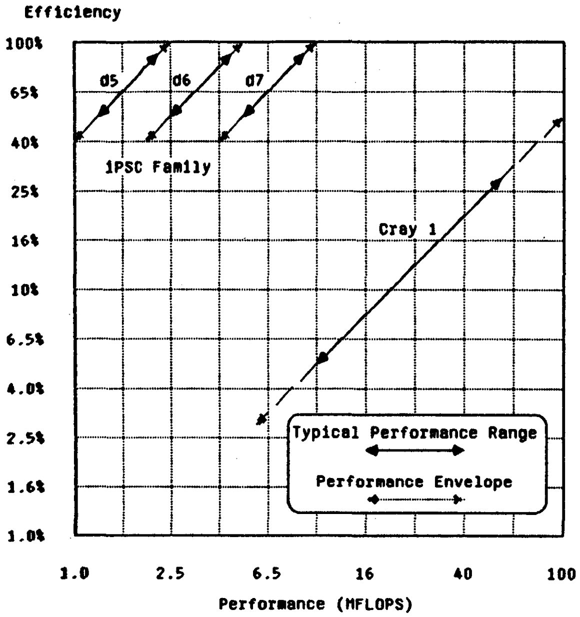
efficiency rate for the Cray 1 on common problems is typically between 5 and 20 percent of peak performance.[5]

A concurrent machine, on the other hand, attempts to share the load equally over a large number of processors, each solving a small portion of the overall task and interacting over very high-speed communication channels. Such division of labor results from the natural concurrency of the problem. The task of any given processor in the system is scaled proportionally to the size of the segment being processed; each processor solves a portion of the outer loop and a portion of the inner loop of the problem. Both vector and scalar performance remain proportional, because the architecture mimics the structure of the problem itself.[6] (See Figure 4.)

Furthermore, a concurrent machine can handle nonnumerical problems such as event-driven simulation[7] and artificial intelligence. This capability is not found in conventional supercomputers, whose architectures are optimized for handling vector calculations.

Historical Perspective

Concurrent or parallel architectures are not new. As early as 1945, computer scientist Vannevar Bush proposed parallel architectures.[8] And John von Neumann, whose ideas led to the sequential architecture used in most computers today, preferred the parallel approach.* But because the unreliability and bulkiness of vacuum tubes made a parallel machine impractical, this approach was not implemented.



The low computational efficiencies typical of today's supercomputers rob them of the high performance promised by their peak MFLOPS ratings. Computational efficiency can be expressed as the ratio of actual performance to peak performance. Vector processors typically achieve only a fraction of their peak performance. Concurrent architectures that achieve high computational efficiency offer significant cost/performance benefits.

Figure 4—Computational efficiency vs. performance

During the 1950s, programs at IBM, the University of Illinois, and elsewhere were set up to develop parallel architectures for numerical computations. Efforts in the 1960s resulted in the construction of several parallel machines, including the Illiac IV.[9] Projects during the 1970s yielded Burroughs Corporation's Parallel Element Processing Ensemble and Goodyear Aerospace Corporation's Staran.

Parallel processing is exhibited in different ways in today's scientific computers. Array processors use several arithmetic elements (adders, multipliers, arithmetic-logic units) to increase performance. Vector processors combine the multiple arithmetic elements of array processors with pipelining techniques and high-speed electronic components for the hundreds-of-MFLOPS performance claimed for these machines.

Potential Power, Economics

Two approaches are being taken to implementing parallel supercomputers. One connects two or more standard supercomputers together. Cray Research has done this with its Cray X-MP series. But the cost of these supercomputers prohibits connecting more than a few of them. In addition, the use of shared memory in these and similar systems limits the number of processors that can be connected.

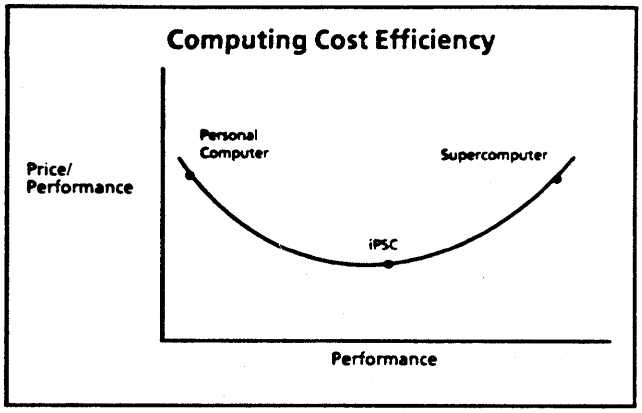
* For a history of parallelism, see R. W. Hockney and C. R. Jesshope, *Parallel Computers* (Bristol, U.K.: Hilger, 1981). For a discussion of parallel processing, see Kai Hwang and Fayé A. Briggs, *Computer Architecture and Parallel Processing* (New York: McGraw-Hill, 1984).

A more practical approach uses a large number of today's most economical computational element, the microcomputer. This approach has been made feasible by the performance and cost advances achieved in recent years through very large-scale integration (VLSI). Because VLSI processors are physically compact and widely available, they are 10 to 100 times more cost effective than the semiconductor technology currently used in conventional supercomputers (Figure 5).

Recently, a number of university research programs have focused on microprocessor-based approaches to large-scale computing. For example, at the California Institute of Technology, Charles Seitz and Geoffrey Fox have developed the hypercube, or "Cosmic Cube," which connects 64 small computers through point-to-point communication channels.[10] Neil Ostlund at the University of Waterloo has built the Waterloop-V2/64, a parallel processor with a 64-node ring architecture.[11] Researchers at Columbia University have produced two concurrent architectures: the DADO machine,[12] which is based on a tree architecture, and the VFPP (Very Fast Parallel Processor), a two-dimensional mesh architecture developed by Norman Christ for computational physics research.[13] At Carnegie-Mellon University, the MMCE (Multi-Micro Computational Engine) is being used for large-scale particle physics computations. The availability of the 8087 floating-point numeric processor and its derivatives has resulted in much of this research activity being focused on Intel microcomputer components and architectures.

Today's supercomputers and array processors are built with high-speed, and sometimes custom-designed, electronic components. Because of limited demand these devices are rarely manufactured in high volumes, and as a result their prices are very high. In contrast, concurrent architectures can be built from standard, off-the-shelf components. Because they are manufactured in large volumes, these components are inexpensive. The result is a dramatic improvement in the ratio of cost to performance.

When the concepts of distributed memory, distributed control, and connected networks are used with concurrent computing architecture, there are few practical limits to the number of processors that can be linked to form a supercomputer system. This is the approach that has been taken with Intel's new iPSC family of high-performance concurrent computers. The iPSC systems employ 32, 64, or 128 processors and have a range of peak performance from 2.5 MFLOPS to 10 MFLOPS, at prices from $150,000 to $520,000.



The high commercial VLSI content fo the iPSC optimizes the performance for the cost over other computational alternatives.

Figure 5—Computing cost efficiency

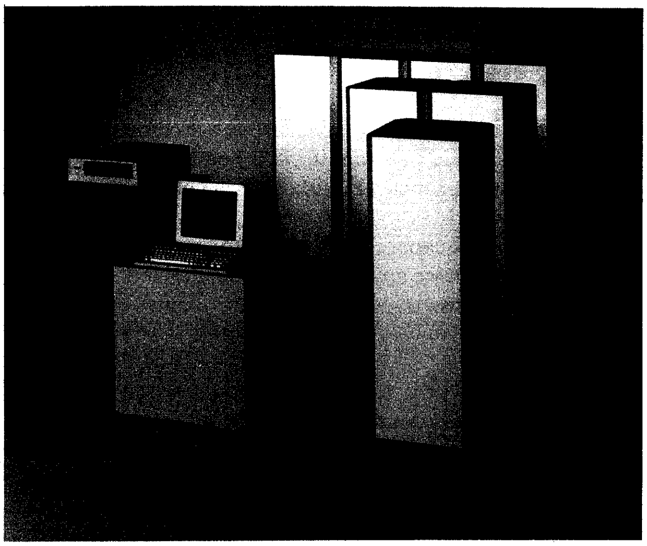
The effect of concurrent processing on scientific computing will be threefold. First, concurrent processing will increase the computer power available for any problem. Second, because concurrent processors will reduce the cost of large-scale computations, they will broaden the range of applications to which computer-based solutions can be applied. Third, the availability of a "focus" machine will accelerate the development of concurrent applications and facilitate the sharing of concurrent systems technology.

The remainder of this paper describes the iPSC systems in more detail.

THE iPSC CONCURRENT COMPUTER

Intel's iPSC is a family of expandable, concurrent computers designed to provide the research community with systems upon which to develop parallel programming techniques, tools, and application programs. Figure 6 shows the iPSC system.

In the iPSC, concurrency is achieved by having an ensemble of loosely coupled independent processors executing portions of a larger computational problem simultaneously. The basic



The iPSC consists of two major system components: the Cube Manager and the Cube. The XENIX-based Cube Manager supports the programming environment and acts as the local host for the Cube. The Cube is the computational element of the system. There are 32, 64, or 128 processing nodes, with a message-based operating system resident on each node.

Figure 6—The iPSC concurrent computer

iPSC system—the *Cube*—consists of 32, 64, or 128 high-performance microcomputers connected to each other via multiple high-speed, point-to-point communications channels. Further, each processor is connected directly to a local host processor (the *Cube Manager*) via a global communications channel. The Cube Manager performs two major functions: it supports the programming environment and serves as the systems manager for the Cube. The Cube Manager permits the iPSC system to operate either as a stand-alone system or as a computational server when networked to a host environment.

System Topology

The iPSC is based on the hypercube topology or interconnection scheme developed by Seitz and Fox at the California Institute of Technology.[10] Caltech's research was conducted under the co-sponsorship of the Department of Energy and the Defense Department's Defense Advanced Research Projects Agency (DARPA). Intel also supported the research by donating microcomputer and memory components.

Researchers at Caltech developed the Caltech hypercube from a concept proposed by Sullivan and Brashkow.[14] A 64-node concurrent computer based on the hypercube topology became operational at Caltech in 1983. After licensing the concept from the university, Intel engaged in further development work, leading to performance improvements and the design of the iPSC system.

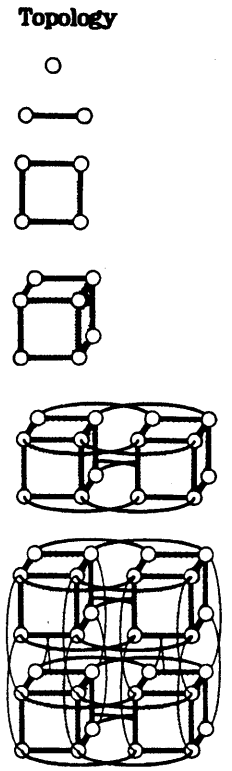
The hypercube is a binary n-cube, also referred to as a *binary hypercube* or *boolean hypercube.* A three-dimensional hypercube is the familiar cube. Higher-dimensioned cubes are built up from this basic structure, with the "dimension" equal to the power of two corresponding to the number of nodes in the cube. Thus, a 32-node cube is a five-dimensional system (2^5) with each node connected to its five nearest neighbors, and so forth. Figure 7 illustrates the hypercube topology.

The hypercube was chosen for a variety of reasons. First, it offers users an option to expand to larger, more powerful systems as needs increase or VLSI-component technology improves. Other approaches, such as shared memory and buses, are limited in the extent to which they can be expanded. The hypercube implementation of the iPSC can be thought of as an open-ended architecture.

Second, the hypercube offers high communications efficiency and communications capabilities that closely match the needs of real problems. That is, the communications among the system elements are optimized for the kind of interactivity that exists in the problems being solved.

Finally, the hypercube can be adapted to suite the size and performance requirements of the problem. A good deal of the research activity in concurrent architectures has focused on the interconnect structure, looking at options such as ring, tree, 2-D mesh, 3-D mesh, and so on. The interconnect structure of the hypercube topology is robust and provides the flexibility to simulate all of these as research into concurrent processing continues. Being a MIMD organization, the hypercube supports both homogeneous and heterogeneous computations.

Dimensions	Nodes	Channels
0	1	0
1	2	1
2	4	4
3	8	12
4	16	32
5	32	80



The hypercube topology provides a rich communication interconnect structure that can be efficiently swapped to lower-dimension topologies. Nodes are connected by point-to-point, dedicated communication channels, thus avoiding contention. The number of communication channels connecting each node is determined by the dimension of the cube; for example, a six-dimension cube connects to six "nearest" neighbors. Communication latency grows only logarithmically for larger-dimension cubes. Communication bandwidth growth is proportional to the size of the cube.

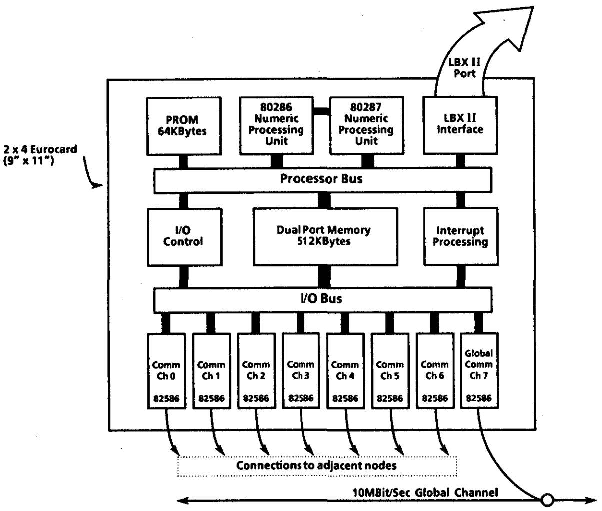
Figure 7—The hypercube topology

System Overview

The initial iPSC family contains three products: the iPSC/d5, a five-dimension machine with 32 computational nodes and 16 megabytes of distributed memory; the iPSC/d6, a six-dimension machine with 64 nodes and 32 megabytes of memory; and the iPSC/d7, a seven-dimension machine with 128 nodes and 64 megabytes of memory. By adding 32 node computational unit users can progressively double the computational power and memory of an iPSC system from 32 to 64 to a maximum of the 128 nodes in the iPSC/d7.

Each microcomputer, or processing node, is made up of an 80286 central processing unit (CPU), an 80287 floating-point numeric processor unit, 512 kilobytes of CHMOS dynamic RAM, and 64 kilobytes of PROM, housed on a single 2 × 4 (approximately 9 × 11 inches) standard Eurocard printed-circuit board. (Figure 8 illustrates the basic node board.)

In addition, each node contains seven point-to-point, bidirectional communications channels and a single global channel. Each channel is controlled by a dedicated communications processor, the 82586 local area network coprocessor.



The importance of VLSI to the design of the iPSC is shown in the more than 11.9 million semiconductor devices contained on each node board. LAN communications coprocessors are used to implement point-to-point communication channels, as well as the Global Channel. The 256K CHMOS DRAMS provide .5 MB of memory per node, and the 80286/80287 CPU/Numeric Processor provides a high-performance computational environment. The iLBX II interface can be used to create two-board nodes.

Figure 8—PSC node board design

These coprocessors move messages between nodes via dedicated bidirectional, point-to-point communications channels with integrated direct memory access to the associated node's RAM.

For enhancement purposes, the local processor-memory bus on each node is accessible via a standard MULTIBUS II iLBX high-speed bus. On the system backplane, the iLBX bus is routed from each even-numbered board slot to the adjacent odd-numbered board slot. The odd-numbered slots may be used for boards that extend the memory or processing capacity of the nodes.

The Cube Manager in the system is an Intel System 310 microcomputer running under the XENIX operating system. It serves as a local host for the Cube and supports program development, applications execution, and diagnostics. It has a 2-megabyte memory, a 40-megabyte Winchester disk drive, and a 320-kilobyte floppy disk drive. In addition to the XENIX operating system, Cube Manager software includes FORTRAN, C, cube control utilities and communications, and complete system diagnostics.

An iSBC 186/51 high-speed communications board provides a global channel to the iPSC system. The iPSC system can also be networked to a mainframe or a superminicomputer by adding a MULTIBUS-based Ethernet TCP/IP interface.

Messaged-Based Interprocess Communications

To coordinate the activities of up to 128 processors, the iPSC uses a message-based operating system resident in each node. This Node Kernel multiplexes the processes that run on that node, provides the system calls that enable processes to send and receive messages, and routes messages as they flow through the hypercube network. The kernel also handles node-to-Cube Manager I/O and process debugging.

Both memory and control are distributed throughout the system. Distributed control means that each node in the system solves its own portion of the larger computational problem semi-independently of other elements in the system. The programmer is responsible for assigning processes to nodes on the basis of the structure of the problem.

The worst-case communication latency for the hypercube is $\log_2 N$; for example, in a 64-node system, the worst case would require a message to pass through six processors. Typically, however, latency is much lower because most applications require only nearest-neighbor interaction. Thus, it is generally necessary only to coordinate the computational results on the edges of each node's data space. For these problems, the larger the computational problem, the better the ratio between computation and communication, and thus the higher the efficiency of the system. The iPSC operating system easily accommodates process-to-process communication, whether the process being communicated with resides in the same node or halfway across the cube.

Low Cost, High Performance

The initial iPSC products will provide a performance range approximately four-tenths that typical of a Cray 1, at no more than one-twentieth the cost. Users report that the Cray 1 typically performs at 10 to 35 MFLOPS on common problems, or at an efficiency rate of about 5 to 20 percent of peak performance. The iPSC typically operates at between 80 and 90 percent efficiency.

The iPSC family is roughly 6 to 24 times as powerful as a DEC VAX-11/780, which costs about $225,000. The price for the iPSC products ranges from $150,000 to $520,000. This reduction in cost—and indeed, the machines themselves—would not have been possible without VLSI technology. As an illustration of the crucial role of VLSI components in the iPSC, consider that each iPSC node board contains 11.9 million silicon devices. Thus, the iPSC/d5, with 32 nodes, contains 380.8 million silicon devices; and the iPSC/d7, with 128 nodes, contains 1.523 billion silicon devices. This level of functionality could not have been achieved in a practical way without the VLSI contribution.

APPLICATIONS

The iPSC is expected to be used in academia, government, and industry. Scientific researchers and computer scientists make up the first two groups. For them, an iPSC system will be the primary vehicle (none is currently available) for research in concurrent computing. The iPSC computer also will be an important research tool for computational physicists and chemists.

The third group of iPSC users will apply the iPSC as a production tool in such applications as circuit simulation and

TABLE I[15]—Examples of problems suitable for concurrent processing

Class of Problems	Examples	Communication Topology
Finite difference equations; finite element equations; partial differential equations	Geophysics, aerodynamics	3D mesh
Statistical	Lattice gauge	4D mesh
	Melting	3D mesh
	Coulomb gas	Ring
Time evolution of 1/r potential	N-body gravity	Ring
Time evolution of general dynamics	Particle motion (sand avalanches)	3D mesh
Fast Fourier transform	Evolution of universe, fluid dynamics	Hypercube
Network simulation	Circuit simulation	Logarithmic graph* such as hypercube
	Neural network	
Isolated	Ray tracing (graphics), data analysis, initial condition study	
Image processing	Analysis of satellite data	Hypercube
Artificial intelligence	Chess	Tree
Event-driven simulation	Industrial, economic, military ("war games")	Logarithmic graph* such as hypercube

* Maximum communication time increases as the log of the number of nodes.

finite-element analysis. The cost effectiveness of the iPSC will stimulate development of applications in the latter group.

Artificial intelligence applications will also benefit from the Intel concurrent processing system. Because concurrent computer architecture is analogous to neurological networks in the human brain (conventional computer architecture is not), computers employing these characteristics may help speed research in this area of artificial intelligence.

Caltech researchers have already put concurrent processing to work in solving a variety of problems in fields such as quantum chromodynamics, structural mechanics, fluid mechanics, high-energy physics, seismology, astrophysics, and computer science. Table I lists several examples of problems that are well suited to concurrent processing.

CONCLUSIONS: A NEW DIRECTION

Research into large-scale computing points to concurrent processing as the most promising technique for increasing supercomputer performance. Concurrent architectures deliver a further benefit in that they can also handle a number of nonnumeric applications that traditional supercomputers cannot accommodate—applications such as artificial intelligence, sorting algorithms, tree searches, and ray tracing.

Nonetheless, significant work remains to be done to develop software for concurrent architectures. Peter Denning, the director of NASA's Institute for Advanced Computer Science, has called attention to this need, pointing out,

> Our research programs are going to have to look at the programming and software problems. We need a lot of work to understand how to make program parts (subroutines) for highly parallel computations and how to plug these parts together to form larger programs. And the programming process will have to include interactive graphic components so programmers can deal with large parallel programs with the aid of pictures. Unless we deal with these issues, we won't be able to take advantage of these machines.[1]

The present state of concurrent computing is similar to the state of real-time control software in the mid-sixties. Before Digital introduced its PDP-8, it was generally felt that such software was very difficult and could be handled only at the largest research centers. Within a few years of the PDP-8's becoming widely available, however, most computer science graduate students knew how to write interrupt handlers for real-time sytems. The widespread availability of a focus machine had dramatically accelerated the accumulation of knowledge by making it easier for researchers to share ideas, exchange programs, and build on each others' work.

I believe that the iPSC family will play a similarly vital role in concurrent computing. The hypercube topology allows expansion to larger systems, fits well with projected directions of VLSI technology, and provides flexibility in the kinds of topologies that can be modeled. The combination of VLSI technology with hypercube topology does indeed provide a new direction for scientific computing: of affordable supercomputers and unlimited increases in performance.

REFERENCES

1. Dallaire, Gene. "American Universities Need Greater Access to Supercomputers." *Communications of the ACM,* 27-4 (1984), pp. 229–298.
2. Seitz, Charles L., and Juri Matisoo. "Engineering Limits on Computer Performance." *Physics Today,* 37-5 (1984), pp. 38–45.
3. Buzbee, B. L. "The Efficiency of Parallel Processing." *Los Alamos Science,* 9, Fall 1983, pp. 71–75.
4. Browne, James C. "Parallel Architectures for Computer Systems." *Physics Today,* 37-5 (May, 1984), pp. 28–35.
5. Davidson, Howard. "Some Predictions on the Performance of Future Supercomputers for Simulation and Control." *IEEE Transactions on Nuclear*

Science, NS-31-1 (1984). Also see Riganati, John P., and Paul B. Schneck. "Supercomputing." *IEEE Computer,* October 1984, pp. 97–112.

6. Charlesworth, Alan E. "An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FP-164 Family." *Computer,* 14: 9 September 1981, pp. 18–27.
7. Jefferson, David. "Implementation of Time Warp on the Caltech Hypercube." *Society for Computer Simulation Distributed Simulation Conference, SCS Simulation Series,* (5)2, 1985.
8. Bush, Vannevar. *Science, The Endless Frontier: A Report to the President.* Washington, D.C.: United States Government Printing Office, 1945.
9. Cocke, J. and Slotnick, D. L. "The Use of Parallelism in Numerical Calculations," IBM Research Memorandum, RC-55, 21, July 1958.
10. Seitz, Charles L. "The Cosmic Cube." *Communications of the ACM.* 28-1 (1985), pp. 22–33.
11. Ostlund, Neil S., and Robert A. Whiteside. "A Machine Architecture for Molecular Dynamics: The Systolic Loop." to be published *Annals of the New York Academy of Science,* 1985.
12. Stolfo, S. J. and Shaw, D. E. "DADO: A Tree-Structured Machine for Production Systems." *AAAI 82.* Carnegie-Mellon University, August 1982. Also see Stolfo, D. P. Miranker and D. E. Shaw, "Architecture and Applications of DADO: A Large-Scale Parallel Computer for Artificial Intelligence." *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Menlo Park, Calif.: IEEE, 1983.
13. Christ, Norman H. and Terrano, A. E. "A Very Fast Parallel Processor." *IEEE Transactions on Computing,* C-33-4 (1984), pp. 344–350.
14. Sullivan, H., and T. R. Brashkow. "A Large-Scale Homogeneous Machine I & II." *Proceedings of the Fourth Annual Symposium on Computer Architecture.* New York: IEEE, 1977, 105–124.
15. Fox, Geoffrey C. and Otto, Steve W. "Algorithms for Concurrent Processors." *Physics Today,* 37-5 (1984), pp. 50–59.

Parallel control algorithms for the reduced-omega-omega^{-1} network

by TSE-YUN FENG* and WEI YOUNG
Ohio State University
Columbus, Ohio

ABSTRACT

Parallel versions of Lee's algorithm for the $N \times N$ ($N = 2^n$) reduced-omega-omega^{-1} network are developed that match the complexities of the parallel versions of the looping algorithm. That is, it runs in $O(\log^2 N)$ time on a completely interconnected computer with N processing elements, in $O(\log^4 N)$ time on a cube-connected or perfect shuffle computer with N processing elements, in $O(N^{1/2})$ time on an $N^{1/2} \times N^{1/2}$ mesh-connected computer, and in $O(k\log^3 N)$ time on a cube-connected or perfect shuffle computer with $N^{1+1/k}$ processing elements.

* Tse-yun Feng is now with the Department of Electrical Engineering, Pennsylvania State University, University Park, Pennsylvania.

INTRODUCTION

For the class of $N = 2^n$ input/output (I/O) rearrangeable switching networks with $2 \log N - 1$ stages of binary switching elements, the looping algorithm had been the only control setting algorithm discovered that ran in O(NlogN) time on a single processor computer. Nassimi and Sahni[1] gave parallel versions of the algorithm running on several types of parallel computers. This greatly improved the running time of the algorithm.

Recently, Lee[2] developed a rather different control algorithm also running in O(N logN) time on a single processor computer for the reduced-omega-omega^{-1} network (which will be defined in the next section). Due to the topological equivalence of the omega network with other types of logN stage blocking networks,[3-5] this algorithm can easily be adapted for some other types of rearrangeable networks (especially the Benes network) with only an extra constant factor involved in the running time. This means the same time complexity.

In this paper, we give parallel versions of Lee's algorithm and compare the results with that of the looping algorithm by Nassimi and Sahni. In so doing, we need the same four types of parallel computer models as described by Nassimi and Sahni[1] and briefly examined next.

The four computer models are considered to be SIMD-type. Assuming each model has M PEs which are indexed from 0 to $M - 1$, and $(i_{t-1} i_{t-2} \dots i_0)$ is the binary representation of i for some appropriate t, we have the following models:

1. *Completely interconnected computer (CIC):* There is a direct connection between each pair of PEs. Therefore, a single communication takes constant time.
2. *Mesh-connected computer (MCC):* PEs are logically considered to be aligned in a k-dimensional array, say $n_{k-1} \times n_{k-2} \times \dots \times n_0$, where n_i is the size of the i-th dimension and $M = n_{k-1} n_{k-2} \dots n_0$. $PE(i_{k-1}, i_{k-2}, \dots, i_0)$ is connected to the PEs at $(i_{k-1}, \dots, i_j \pm 1, \dots, i_0)$, where $0 \leq j < k$, if they exist.
3. *Cube-connected computer (CCC):* PE(i) is connected to $PE(i^{\bar{j}})$, for $0 \leq j < t$ and $M = 2^t$, where i^j is the number whose binary representation is that of i with the j-th bit complemented. That is, if $i = i_{t-1} i_{t-2} \dots i_0$, then $i^{\bar{j}} = i_{t-1} \dots \bar{i}_j \dots i_0$ where $\bar{i}_j$ is the complement of i_j.
4. *Perfect shuffle computer (PSC):* PE(i) is connected to $PE(i^{\bar{0}})$, PE(s(i)), and PE(u(i)), where $s(i) = s(i_{t-1} i_{t-2} \dots i_0) = i_{t-2} \dots i_0 i_{t-1}$ is the so-called *shuffle,* and $u(i) = u(i_{t-1} i_{t-2} \dots i_0) = i_0 i_{t-1} \dots i_1$ is the so-called *unshuffle.*

A PARALLEL VERSION ON A CIC

To avoid confusion, we will follow most of Lee's notations and examples[2] to illustrate our algorithms.

A Brief Review of Lee's Algorithm

A reduced-omega-omega^{-1} network (omega^{r}omega^{-1}) is an omega network without the last stage of switching elements (SEs), followed by a reverse omega network (omega^{-1}) which is just an omega network with the original roles of inputs and outputs exchanged. If the number of IOs should be emphasized, we use omega$_N$ to denote an omega network with N inputs and N outputs. In the example of omega$_8^{-1}$ shown in Figure 1, each rectangle stands for a switching element that is able to set its inputs straight through or cross-over.

In general, an n-stage omega network can be represented as $\text{omega}_N = sE^{n-1} sE^{n-2} \dots sE^0$, and hence $\text{omega}_N^{-1} = E^0 uE^1 u \dots E^{n-1} u$, where s and u are the shuffle and the unshuffle, respectively, and each E^i denotes a switching element stage. For an omega$_N^{-1}$, let $A^i = (a_0^i, \dots, a_{N-1}^i)$ be the input permutation of the stage E^i. Also, let

$$A_{0,K}^i = \{a_k^i | 0 \leq k < 2^n\},$$
$$A_{j,k}^i = A_{j-1,2k}^i \, U A_{j-1,2k+1}^i,$$
where $0 \leq k < 2^{n-j}$, $1 < j < n + 1$, and
$$B_k^i = \{b_{k,m}^i | 0 \leq m < 2^i\},$$
where $b_{k,m}^i = a_{k+m \times 2^{n-j}}^i$, $0 \leq k < 2^{n-i}$ and $0 < i < n + 1$.

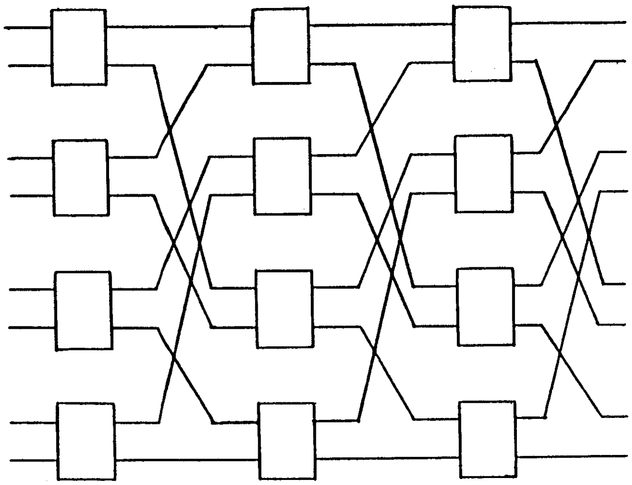


Figure 1—An omega^{-1} network where n = 3

(See Figures 1–3 for illustrations of these notations.) It is easy to see that $B^i_k = A^0_{i,k}$.

Define an input permutation $A^0 = \left(a^0_0, a^0_1, \ldots, a^0_{N-1}\right)$ to be omega^{-1}-passable if $A^n = A^0(\text{omega}^{-1})$, where $A^n = (0, 1, \ldots, N-1)$. Lee's Theorem 1 (omega^{-1}-passability) states that A^0 is omega^{-1}-passable iff for $1 \leq j < n$ and $0 \leq k < 2^{n-j}$, $A^0_{j,k}$ forms a "complete residues system modulo 2^j" (CRS(mod 2^j)).[2] Therefore, we have a way to check whether a given input permutation is omega^{-1}-passable. Now, if a given input is indeed omega^{-1}-passable, a very simple and essentially parallel (relative to a stage) control scheme for the omega^{-1} part applies. We simply set each switching element of the i-th stage according to the i-th bit of the binary representation of its upper input; that is, set the switch to *straight* if the i-th bit is "0" and to *cross* if the i-th bit is "1". Based on the claimed rearrangeability of the omega^r omega^{-1}, one can conclude that the control algorithm of the first part must convert every input permutation into an omega^{-1}-passable one.

Because an omega is the reverse of an omega^{-1}, we can still use the notations for omega^{-1} on omega, except that we have to switch the roles of input and output. The relations among As and Bs still hold. We know that $B^i_k = A^0_{i,k}$, hence the first part-control algorithm should make each B^i_k a CRS(mod 2^i), according to the omega^{-1} passability. Lee[2] shows us that

$$\left(b^i_{2k,m}, b^i_{2k+1,m}\right)E^i = \left(b^{i+1}_{k,m}, b^{i+1}_{k,m+2^i}\right),$$
where $0 \leq m < 2^i$, and
$$B^i_{2k} \, U B^i_{2k+1} = B^{i+1}_k,$$
where $0 \leq k < 2^{n-(i+1)}$ and $0 \leq i < n$.

This means that a stage of switching elements (E^i) must partition each B^{i+1}_k, a CRS(mod 2^{i+1}), into two (CRS(mod 2^i))s, B^i_{2k} and B^i_{2k+1}.

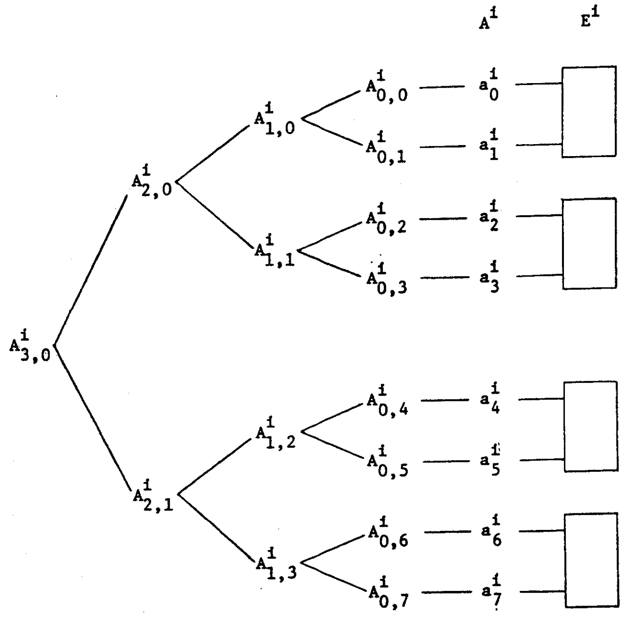


Figure 2—Partitions of As where n = 3

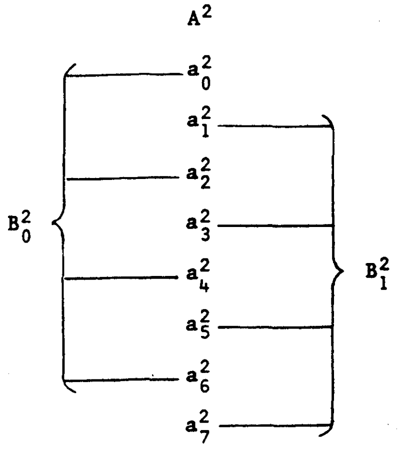


Figure 3—An A and Bs where i = 2 and n = 3

Lee's Lemma 4 describes a sequential process to set the switching elements in a stage.[2] An example is shown in Figure 4. The set of inputs of a stage is $B^3_k = (7, 5, 3, 4, 0, 1, 6, 2)$ (obviously a CRS(mod 2^3)). As the switches are set, we find two (CRS(mod 2^2))s, $B^2_{2k} = (7, 4, 1, 6)$ and $B^2_{2k+1} = (5, 3, 0, 2)$.

As the process is done for each B^i_k of stages from the first to the last stage of the omega^r omega^{-1} network, any input permutation is converted into an omega^{-1}-passable input for the second half. Therefore, the whole algorithm completes the desired connections among input and output terminals correctly.

The Parallel Version

From the discussion above, it is clear that the control scheme of the second part is essentially parallel to a stage, and

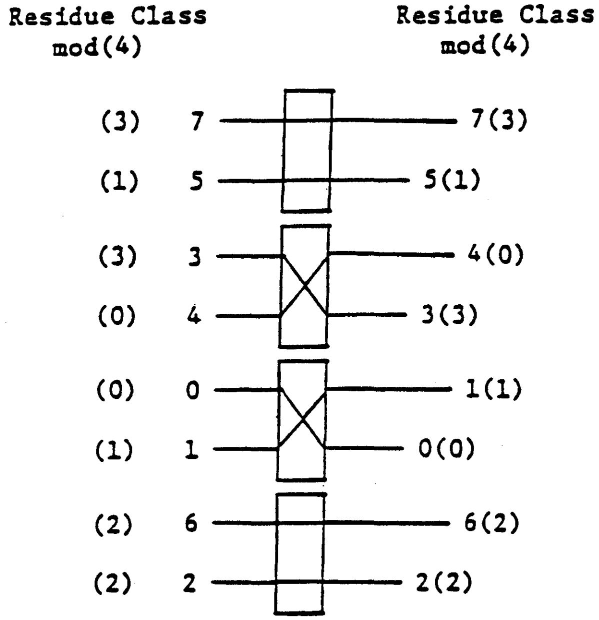


Figure 4—Partition of a B into CRSs[5]

that the complexity is O(log N), which is the order of the number of stages. However, in the first part, the time to set up a single stage is O(N). With log N − 1 stages, the total time is O(N log N). Adding the time for the second part, we still have O(N log N). This is of the same order of time complexity as the serial looping algorithm. Our algorithm will make the process of the first part parallel.

Let X and Y be finite sets with no elements in common. Let E be a set of pairs of the form [x, y], where x is in X and y in Y. The elements of X and Y are called *nodes,* and the pairs [x, y] in E are called *edges.* The resulting combination is called a *bipartite graph* and is denoted by ⟨X, E, Y⟩. If e = [x, y] is an edge, then x and y are called the *nodes* of the edge e, and we say that e *joins* x and y and that x and y *meet* e. The *degree* of a node is the number of edges which the node meets. A bipartite graph is *regular* if every node has the same degree. A set of edges is a *matching* of a bipartite graph provided that no two of the edges have a node in common. A matching is called *perfect* if every node of the graph meets exactly one edge in the matching. It is known that a regular bipartite graph of a positive degree has a perfect matching.

A closer look at the process of partitioning B_k^{i+1} shows that there are exactly two members of B_k^{i+1} in each residue class relative to 2^i. If we let X be the set of pairs of inputs of switching elements, Y be the set of residue classes relative to 2^i, and E be the set of edges of the form [x, y], where x is a pair with one of its members in the residue class y mod 2^i, then ⟨X, E, Y⟩ is a regular bipartite graph of degree 2. Figure 5 shows the bipartite graph of the example in Figure 4. As we can see, there are exactly two edges incident to every node. Since both B_{2k}^i and B_{2k+1}^i are CRS(mod 2^i), the process of partitioning amounts to finding a perfect matching for the corresponding bipartite graph of B_k^{i+1}. Once found, the rest of the edges will automatically form another perfect matching due to the regularity of the bipartite graph of degree 2. Thus, all we have to do is ensure that members of a matching come out on the upper outputs of all the switching elements which constitute B_{2k}^i, and the set of all the lower outputs will naturally be B_{2k+1}^i.

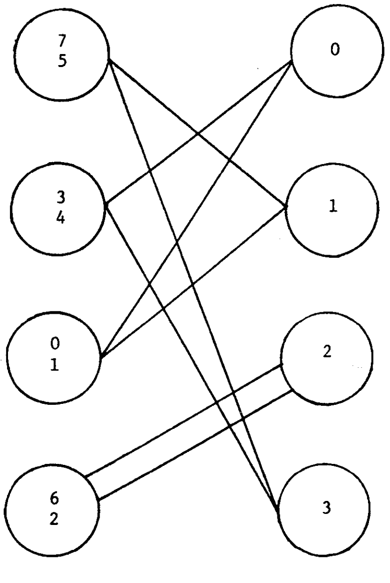


Figure 5—A bipartite graph for Figure 4

We first construct a regular bipartite graph of degree 2 for B_k^{i+1} as discussed above. Note that an input a_t must be in the same matching including the input a_{m+1}, or a_{m-1} of the switching element whose other input a_m is equivalent to a_t relative to 2^i (or $a_m \equiv a_t \bmod 2^i$) because one input of a switching element must go to B_{2k}^i and the other must go to B_{2k+1}^i. Clearly a_t and a_m cannot be in the same partition. Now we can easily build links among inputs of the same partition by linking a_t to either a_{m-1} or a_{m+1}. We then let the identities of B_{2k}^i and B_{2k+1}^i propagate through the links inside individual partitions at an exponential rate. At the end of this step, the control settings are essentially established, because each input already "knows" to which partition (B_{2k}^i or B_{2k+1}^i) it belongs, and hence to which output (upper or lower) it should go.

Assume that a CIC with N PEs is at our disposal. In each PE(i) where $0 \le i < N$, we have a storage DST(i) holding the index of the final output terminal to which the input terminal i should be connected in each stage of the process; a BK(i) specifying to which B_k^i the DST(i) belongs; an A(i) with the two subfields *first* and *second,* denoted by A(i).1st and A(i).2nd, holding indices of the PEs whose DSTs are in the same residue class that PE(i) stands for; a SW(j, i) where $0 \le j < 2 \log N - 1$ storing the settings of the corresponding switching elements in the j-th stage; an L(i) holding the link to a member of the same CRS(mod 2^i) in the partition of a CRS(mod 2^{i+1}); and some temporary storages such as R(i), C(i), D(i), etc., as needed for processing. In the following algorithm, $x_{n-1}x_{n-2}\ldots x_0$ is the binary representation of x,

```
Line            Procedure Control_omega$^r$

 1  BK(i) = 0; DST(s(i)) ← DST(i)
      //inputs in $B_0^n$ and perform first shuffle//
 2  for k = n − 1 to 1
      //partition starts//
 3      R(i) = DST(i) mod $2^K$ + BK(i) × $2^k$
          //compute index of the PE for DST(i) mod $2^k$//
 4      A(R(i)).1st or A(R(i)).2nd ← i
          //each PE identify itself to the proper PE//
 5      L(A(i).1st) ← (A(i).2nd$^{\bar{0}}$   ($i < 2^{n-1}$)
        L(A(i).2nd) ← (A(i).1st$^{\bar{0}}$   ($i < 2^{n-1}$)
          //establish links inside each partition//
 6      C(i) = i
          //initialize a temporary storage//
 7      for j = 1 to k
 8          (D(L(i)), TEMP(L(i))) ← (C(i), i)
 9          L(TEMP(i)) ← L(i)
10          C(i) = min{C(i), D(i)}
11      endfor //propagate identities of partitions//
          //partition ends//
12      SW(n − 1 − k, i/2) ← $(C(i))_0$   ($i_0 = 0$)
          //control setting//
13      DST($i^{\bar{0}}$) ← DST(i)   ($i_0 \neq (C(i))_0$)
14      BK(i) = i mod $2^{n-k}$
15      (DST(s(i)), BK(s(i))) ← (DST(i), BK(i))
          //update inputs for the next stage//
16  endfor
      //procedure ends//
```

and $x^{\bar{0}} = x_{n-1}x_{n-2}\ldots\bar{x}_0$ (the bar means *complement*) is the number obtained by complementing the least significant bit of x. The symbol " = " is an assignment inside a PE in the usual sense of programming languages, while the symbol ← is an assignment from one PE to another through interconnections. Assume that the input permutation is already in each DST(i) before the algorithm starts.

In partitioning a B^i_k, we actually need 2^{i-1} processors because there are only that many residue classes. There are also $2^{n-i}(B^i_k)$s in the i-th stage. Therefore, only $2^{n-1} \times 2^{i-1} = 2^{n-1} = N/2$ processors are needed in any stage. Also, since there are only N/2 switching elements in a single stage, this algorithm only needs N/2 processing elements. But for the sake of comparing our results with the parallel versions of the looping algorithm, we now use N processors. An N/2 processor version with the same order of time complexity will be given later. With this notion in mind, let us analyze the above algorithm.

Line 1 achieves the first shuffle and initializes the partition to which each input belongs. This step takes O(1) time. To process B^{t+1}_m, we use the m-th group of PEs. For example, we use PE(0), PE(1), ..., PE($2^t - 1$) to process B^{t+1}_0, PE(2^t), PE($2^t + 1$), ..., PE($2^{t+1} - 1$) for B^{t+1}_1, ..., etc. In general, we use the group PE($m2^t$), PE($m2^t + 1$), ..., PE($m2^t + 2^t - 1$) to process B^{t+1}_m.

Line 3 computes the address of the target PE with the help of BK(i) which specifies the correct PE group and DST(i) which specifies the residue class in each group. This step also takes O(1) time. In line 4, each PE identifies itself to the appropriate residue class-processing PE. Since there are exactly two members of B^{t+1}_m in each residue class mod 2^t, PE(i) where $i < 2^{n-1}$ should receive two identifications in this step. With a proper arbitration mechanism, the two identifications should be put into A(i).1st and A(i).2nd in no more than three communication cycles. Hence, this step also takes O(1) time.

Line 5 establishes links inside each partition. As mentioned earlier, every input a_p must be in the same partition including the input a_{q-1}, or a_{q+1} of the switching element whose other input a_q is equivalent to $a_p \bmod 2^t$. Hence, we complement the least significant bit of the index of the underlying PE and set up the link (L(i)). It is clear that all of the links (L(i)s) are distinct. Therefore, when we traverse along these links inside each partition, we get several cycles. For example, in Figure 6, (7, 3, 1), (5, 0, 3), (2), and (6) are the cycles. We will have (7, 4, 1, 6) and (5, 0, 3, 2) as partitions at the end of this algorithm. It is obvious that line 5 takes O(1) time.

Lines 6 through 11 propagate the identity of a partition to its members by letting the smallest index ripple through each cycle in each iteration, exponentially. For example, in Figure 6, the cycle (7, 4, 1) should have the identification 0 because 7 is in PE(0), (5, 3, 0) should have the identification 1 because 5 is in PE(1), and for (2) and (6), the identifications are 6 and 7, respectively. To achieve this, we first initialize C(i) to i in line 6, then have lines 8 and 9 update L(i) so that it points to the PE which is 2^j units away from PE(i) in the partition (in the sense that two directly-linked nodes are one unit away), and then put the index of the PE which is 2^{j-1} units away from PE(i) into D(i). Line 10 takes the minimum of C(i) and D(i). This makes C(i) the smallest index among all PEs which are no greater than 2^j units away from PE(i) because, from the previous iteration, C(i) is the smallest among those PEs which are originally no greater than 2^{j-1} units away from PE(i), and D(i) is actually C(m) where PE(m) is 2^{j-1} units away from PE(i). After k iterations, C(i) holds the smallest index among those PEs which are 2^k units away from PE(i), but a cycle can be no more than 2^k units long for a B^k_r. Hence, the identity of each partition will have been distributed through after these lines. The time complexity of lines 7 through 11 is clearly O(k), since lines 8, 9, and 10 take O(1) time and result in k iterations.

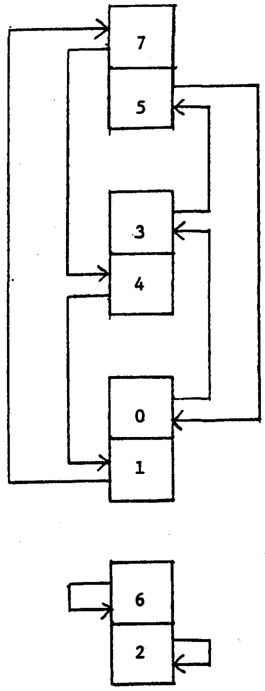


Figure 6—Links in two partitions

Line 12 stores the control settings according to the least significant bit of C(i)s for even-numbered indexed PEs, having only N/2 switching elements to set. This step takes O(1) time. Line 13 interchanges the two inputs if the switching element they belong to is set to *cross*. Line 14 computes the index of the partition in which the DST(i) should be in the next stage. Line 15 moves the processed permutation with partition information to the next stage through a shuffle. Again, each line (13, 14 and 15) takes O(1) time. Since lines 1 through 16 have to be performed n − 1 times, and for each iteration, lines 7 through 11 take 0(k) where $1 < k < n$ while the rest of the lines take constant time, the time complexity of Control_omega^r is $O(n^2) = O(\log^2 N)$ (due to the fact that $(n-1) + (n-2) + \ldots + 1$ is $O(n^2)$, the same as the parallel version of the looping algorithm on a CIC with N processors).

In the following procedure, we give a variant of Control_omega^r which uses a CIC with N/2 processing elements as mentioned earlier. Essentially, every step of this variant is the same as in the Control_omega^r procedure except that we have all of the information that was originally in two PEs in a single PE. This may take twice as much time for some steps, but as far as time complexity is concerned, the variant remains $O(n^2)$. Because of the similarity of the variant to Control_omega^r, we will only list the algorithm without further discussion.

In each PE, we need the same types of storage units performing the same functions except that we must have two copies for each of them. This is achieved because $j = 0, 1, A(i, j), DST(i, j), BK(i, j)$, and $L(i, j)$ where both $A(i, j)$ and $L(i, j)$ have two subfields (*first* and *second*) and, of course, some temporary storages which will appear as they are needed. We also need two extra storage units, $idx(i, 0)$ and $idx(i, 1)$, to hold the indices of the corresponding PEs for the purpose of performing the shuffles and computing $BK(i, j)$ easily.

Procedure Control_omega^r(N/2)

```
$BK(i, j) = 0; idx(i,j) = 2 \times i + j$
$E(i, j).1st = s(idx(i, j))/2; E(i, j).2nd = (s(idx)i, j))_0$
$DST(E(i, j).1st, E(i.j).2nd) \leftarrow DST(i, j)$
for $k = n - 1$ to 1
  //partition//
    $R(i, j) = DST(i, j) \bmod 2^k + BK(i, j) \times 2^k$
    $A(R(i, j), 0)$ or $A(R(i, j), 1) \leftarrow (i, j)$
    $L(A(i, j).1st, A(i, j).2nd) \leftarrow (A(i, \bar{j}).1st, (A(i, \bar{j}).2nd)^{\bar{0}})$
    $(C(i, j).1st, C(i, j).2nd) = (i, j)$
    for $m = 1$ to $k$
        $(D(L(i, j)), TEMP(L(i, j))) \leftarrow (C(i, j), (i, j))$
        $L(TEMP(i, j)) \leftarrow L(i, j)$
        $C(i, j) = \min_{first\_field}\{C(i, j), D(i, j)\}$
    endfor
      //partition ends//
    $SW(n - 1 - k, i) = (C(i, 0).2nd)_0$
    $DST(i, 0) \longleftrightarrow DST(i, 1)((C(i, 0).2nd)_0 \neq 0)$
      // $\longleftrightarrow$ means interchange//
    $BK(i, j) = idx(i, j) \bmod 2^{n-k}$
    $(DST(E(i, j)), BK(E(i, j))) \leftarrow (DST(i, j), BK(i, j))$
endfor //end of the procedure//
```

To complete the routing for the whole network, we let the output from the omega^r pass through the omega^{-1} controlled by the scheme described above. We list it as follows:

List Procedure Control_omega^{-1}

```
1 for $k = 0$ to $n - 1$
2     $SW(k = n, i/2) \leftarrow (DST(i))_k \quad (i_0 = 0)$
3     $DST(i^{\bar{0}}) \leftarrow DST(i) \quad (i_0 \neq (DST(i))_k)$
4     $DST(s(i)) \leftarrow DST(i)$
5 endfor
```

In this procedure, line 2 sets the switching elements according to the k-th bit of the upper input. Line 3 interchanges two inputs of a switching element if it is set to *cross*. Line 4 shuffles the permutation to the next stage. Each of these lines clearly takes $O(1)$ time. We have a total of n iterations from lines 1 to 5; hence, the time complexity is $O(n) = O(\log N)$. Adding this to the Control_omega^r, we have a time complexity of $O(n^2) = O(\log^2 N)$ for the omega^r omega^{-1} network on a CIC. Figure 7 shows the complete result of the input permutation in Figure 4 after going through the omega^{r}omega^{-1} network.

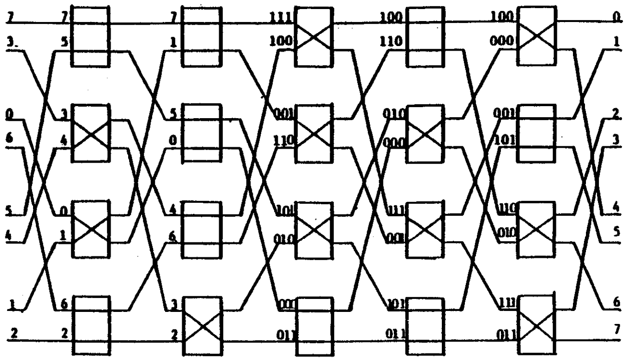


Figure 7—A complete control setting

PARALLEL VERSIONS ON OTHER MODELS

When we use models other than the CIC, a PE is connected only to *some* other PEs, but not to *every* other PE. Lines 1, 4, 5, 8, 9, 12, 13 and 15 may take more than $O(1)$ time, but generally the functions of all of these lines can be achieved through sorting. For example, lines 1 and 15 can still be done in constant time on PSCs, but not on MCCs or CCCs. However, for MCCs and CCCs, we can set up a 3-tuple $\langle s(i), DST(i), BK(i) \rangle$ in each PE(i), then sort the 3-tuples on the first field (i.e., on s(i)). After the sorting, every 3-tuple will be in the correct place. Line 13 can still be done in constant time because PE(i) remains connected to PE($i^{\bar{0}}$) in any case, according to the definitions. Line 12 can be done by first setting up $\langle i/2, (C(i))_0 \rangle$ for each PE which is even-number indexed, and $\langle$infinity, infinity$\rangle$ for those which are odd-number indexed, and then performing a sort on the first field. For line 8, we set up $\langle L(i), C(i), i \rangle$s and a sort on the first field. This step gives the address of the PE to which a PE should send a message in the next step. For line 9, we then set up $\langle TEMP(i), L(i) \rangle$s and a sort on the first field to complete the entire communication step. For line 5, since only the first half of the PEs are sending out messages at a given time, we use two steps. First, we set up $\langle A(i).1st, (A(i).2nd)^{\bar{0}} \rangle$ for each PE(i) where $i < 2^{n-1}$, and $\langle -1, -1 \rangle$ for the rest of the PEs, and then a sort on the first field. This step actually pushes all of the meaningful records down to the last 2^{n-1} PEs. We now change $\langle -1, -1 \rangle$s to $\langle A(i).2nd, (A(i).1st)^{\bar{0}} \rangle$s for PE(i)s where $i < 2^{n-1}$, and leave those records in the rest of the PEs unchanged. A sorting then completes line 5. For line 4, we set up record $\langle R(i), i \rangle$ for each PE(i), then a sort on the first field. Since, for each PE(i) where $i < 2^{n-1}$, we should have two indices, and since PE(i) and PE($i^{\bar{0}}$) are directly connected, we let every PE(i) with $i_0 = 1$ send its record to PE($i^{\bar{0}}$) and change its record to $\langle$infinity, infinity$\rangle$. A sorting then pushes all the meaningful records up to the first 2^{n-1} PEs, hence completing line 4. By the same reasoning, lines 2, 3 and 4 of the Control_omega^{-1} can also be accomplished by sortings.

From the above discussion, we know that every step which requires direct communication among PEs on CICs can be achieved by no more than three sortings on other models. Therefore, the time complexity of the control algorithm on

MCCs, CCCs or PSCs is simply the time on CICs multiplied by the time needed to sort on individual models. As summarized by Nassimi and Sahni, the best sorting method on CCCs of PSCs with N PEs takes $O(\log^2 N)$ time.[1] Hence, the Control_omega^r omega^{-1} takes $O(\log^4)$ time. Sorting on CCCs or PSCs with $N^{1+1/k}$ PEs takes $O(k \log N)$ time. Therefore, our algorithm takes $O(k \log^3 N)$ time. Sorting on MCCs with $N^{1/2} \times N^{1/2}$ PEs takes $O(N^{1/2})$ time, but due to the decreasing size of each partition to be sorted, and the special $N^{1/2} \times N^{1/2}$-mesh construction, we need only $O(N^{1/2})$ to run the entire Control_omega^r omega^{-1} as computed by Nassimi and Sahni.[1]

CONCLUSION

We have shown that Lee's algorithm can be parallelized and have discussed an algorithm developed to run on several types of models with different time complexities. Compared with the parallel versions of the looping algorithm, every parallel version of Lee's algorithm is of the same order of time complexity as its counterpart of the looping algorithm on the same type of model.

Control_omega^r omega^{-1} can also run on models with N/2 processing elements. Although some steps may take twice as much time, other steps, such as interchanging records between two adjacent processing elements in the N processor models, are simply operations within individual processors. As such, they require no communication among processors, and hence can be done faster. Therefore, run times may not change significantly, but in any case, the time complexity is still of the same order. Thus, models using fewer processors are viable alternatives that may prove useful when hardware or other considerations are involved.

REFERENCES

1. Nassimi, D., and S. Sahni. "Parallel Algorithms to Set Up the Benes Permutation Network." *IEEE Transactions on Computers.* C-31-2 (1982), pp. 148–154.
2. Lee, K. Y. "On the Rearrangeability of a 2 log N − 1 Stage Permutation Network." *Proceedings of the International Conference on Parallel Processing*. Silver Spring, Md.: IEEE Computer Society Press, 1981, pp. 221–228.
3. Wu, C., and T. Feng. "The Reverse-exchange Interconnection Network." *IEEE Transactions on Computers.* C-29-9 (1980), pp. 801–811.
4. Wu, C., and T. Feng. "On a Class of Multistage Interconnection Networks." *IEEE Transactions on Computers.* C-31-8 (1980), pp. 694–702.
5. Parker, D. S., Jr. "Notes on Shuffle/Exchange-type Switching Networks."*IEEE Transactions on Computers.* C-29-3, (1980), pp. 213–222.

SUGGESTED READINGS

Brauldi, R. A. *Introductory Combinatorics.* Amsterdam: North Holland, 1979.

Lawrie, D. H. "Access and Alignment of Data in an Array Processor." *IEEE Transactions on Computers,* (1975), pp. 1145–1155.

Lev, G., N. Pippenger, and L. G. Valiant. "A Fast Parallel Algorithm for Routing in Permutation Networks." *IEEE Transactions on Computers,* C-30-2 (1981), pp. 93–100.

Opferman, D. C. and N. T. Tsao-wu. "On a Class of Rearrangeable Switching Networks." *Bell Systems Technical Journal,* 50-5 (1971), pp. 1579-1618.

Shiloach, Y. "An O(log n) Parallel Connectivity Algorithm." *Journal of Algorithms* 3 (1982), pp. 57–67.

Interconnecting off-the-shelf microprocessors

by HUMOUD B. AL-SADOUN, O. A. OLUKOTUN, and T. N. MUDGE
University of Michigan
Ann Arbor, Michigan

ABSTRACT

This paper outlines the design and analysis of a crossbar-type interconnection network that can be used with off-the-shelf microprocessor components to construct a four-processor/eight-memory module multiprocessor system. The only custom component is an interconnection (ICN) chip presently being fabricated by the authors. The ICN chip integrates 4 × 8 crosspoints for a 3-bit bus slice together with related priority circuitry into a single component. System design using the ICN chip is illustrated with the Intel 8086 family of microprocessor components. An analysis of the resulting system is presented. The degradation of performance resulting from the memory interference associated with multiprocessors is shown to be small compared to that resulting from the setup time required by the interconnection logic.

INTRODUCTION

The use of multiprocessors in appropriate situations can improve the performance of a wide variety of computing tasks, particularly with respect to speed and reliability. Several currently available microprocessor families provide VLSI components that support the construction of multiprocessor systems. Typically, these components are intended for shared-bus MIMD multiprocessors. Such systems are composed of a number of processors which share a common memory by means of a single shared bus and which may possibly have local memory. The shared bus can be a bottleneck that offsets the advantage of having multiple processors if their combined request rate to the shared memory exceeds the bus bandwidth. In this paper we outline the design of an integrated circuit, the ICN (interconnection) chip, that simplifies the construction of multiprocessors that are not constrained to share memory through a single bus. Specifically, the ICN chip together with off-the-shelf microprocessor support chips makes possible the construction of a system in which four processors share eight memory modules through a crossbar interconnection. Individual copies of the chip integrate the necessary logic for a 3-bit bus slice together with related priority circuitry. The ICN chip is presently being fabricated by the authors and is the successor to two earlier prototypes.[1,2] In principle the concept can easily be extended to systems with N processors sharing M memories, subject only to the constraint of pin limitations imposed by the technology used to package the ICN chip. By organizing the interconnection logic as a stack of slices, there can be considerable replication of priority logic and crosspoint selection logic; however, reducing the number of components, rather than gates, and making the components easy to use are more important considerations from a systems design viewpoint. This is evident from the typical replication of address logic that results from organizing memories from slice components.

If a high bandwidth connection to memory is required in a multiprocessor, there are two major advantages to using a crossbar organization compared to multistage alternatives such as Delta networks, cube networks or Banyan networks.[3] First, there is the relative ease with which crossbars can be controlled in MIMD mode. Second, there is the absence of intra-network interference that arises with multistage networks. These advantages are bought at the expense of gate complexity. In particular, the gate complexity of the crossbar is $O(n^2)$ if $N = M = n$, and that of the Delta, cube, or Banyan is $O(n \log_2 n)$; however, in terms of VLSI layout the space complexity is closer to $O(n^2)$ in both cases if logic gates are discounted,[4,5] and higher if priority and related logic are considered.[1] Thus, for systems with less than a few dozen processors operating in MIMD mode the advantages of a crossbar connection outweigh its disadvantages.

The remainder of this paper is organized as follows. The next section describes the operation of a crossbar constructed from ICN chips and Intel support chips. The section following the next section develops an analytical model for the performance of multiprocessor systems constructed using ICN chips. Closing remarks are presented in the conclusion.

A CROSSBAR-BASED MULTIPROCESSOR

As noted, ICN chips can be used to construct a crossbar to interconnect four processors and eight memories. The crossbar allows simultaneous connections between the processors and the memories. The resulting multiprocessor is diagrammed in Figure 1. The crossbar is designed so that it interfaces directly with Intel 8086 microprocessors augmented by several support chips; in particular, Intel 8289 bus arbiters are used to multiplex processors onto multimaster system buses and avoid contention problems between bus masters. Each memory port can be regarded as a multimaster system bus, and each processor can be regarded as a bus master. The crossbar requires four 8289s to control the access of the four processors to the eight memories, and since each multimaster system bus is made up of 40 signal lines, the crossbar requires 14 ICN slices.

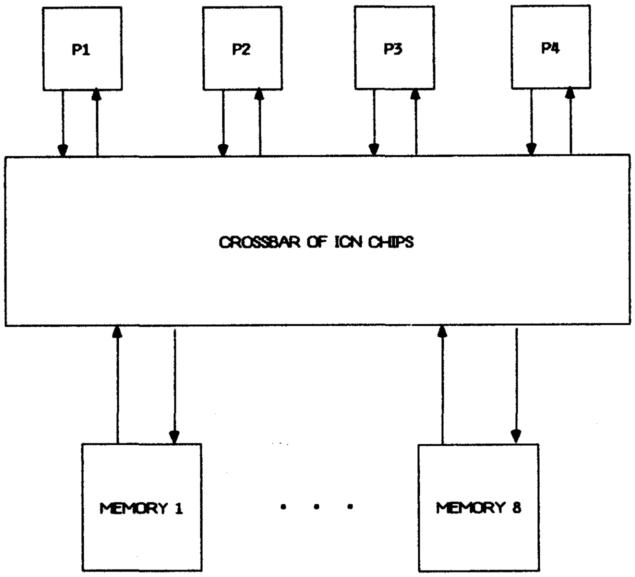


Figure 1—Crossbar-based multiprocessor

Interface between Processor and Crossbar

In common with most microprocessors, the 8086 lacks the capability of requesting bus access and recognizing bus grants from a multimaster bus. Therefore it is necessary to have extra logic to generate the necessary signals for sending bus requests and receiving bus grants. This extra logic comes in the form of an Intel 8289 bus arbiter, one of which is associated with every bus master (processor) in the multiprocessor. These bus arbiters are all synchronized by BLCK, the common bus clock.[6] (See Figure 2 for further details of each of the four processors of Figure 1.)

The 8086 is unaware of its attached arbiter's existence and therefore issues commands as though it had exclusive use of the system memory. The 8289 monitors its 8086's status lines (S2–S0) to detect the beginning of a bus cycle. At the beginning of the bus cycle the bus controller (8288), which also monitors the 8086's status lines, generates an ALE (address latch enable) signal to latch the address information from the 8086. If the processor is in control of the bus (i.e., it has requested and received a memory), it enables the outputs of the bus controller (8288) and the address latches (8283), and the bus cycle continues as usual. If the processor does not have control of the bus, the arbiter forces the outputs of the 8288 and the address latches into their high impedance state. The 8288 in turn forces the outputs of the data transceivers (8287) into their high impedance state. At the same time the clock generator (8284) is prevented from sending a ready signal to the 8086, which forces the 8086 to enter its wait state after T3 of the bus cycle in progress (bus cycles have four subcycles T1, T2, T3 and T4). Once the arbiter is granted bus access, it enables the outputs of the 8288 and the address latches. The addressed memory port returns an acknowledge signal to the clock generator when the data transfer is complete. This acknowledge signal causes the clock generator to send a ready signal to the 8086. The 8086 then exits its wait state and completes the bus cycle in progress.

When an arbiter detects the beginning of a bus cycle and does not have control of the bus, it proceeds to request the bus by activating its BREQ (bus request) line. The BREQ line from each arbiter is fed into the crossbar. Our application makes use of the three highest bits of address to determine which of the eight memories the processor is requesting. Therefore, it is necessary that the address be latched by the crossbar's internal latches and that the connections be established within the crossbar before BREQ is activated by the arbiter (see Figure 3). This sequence of events is ensured by setting the arbiter in resident bus mode (RESB pin high). This

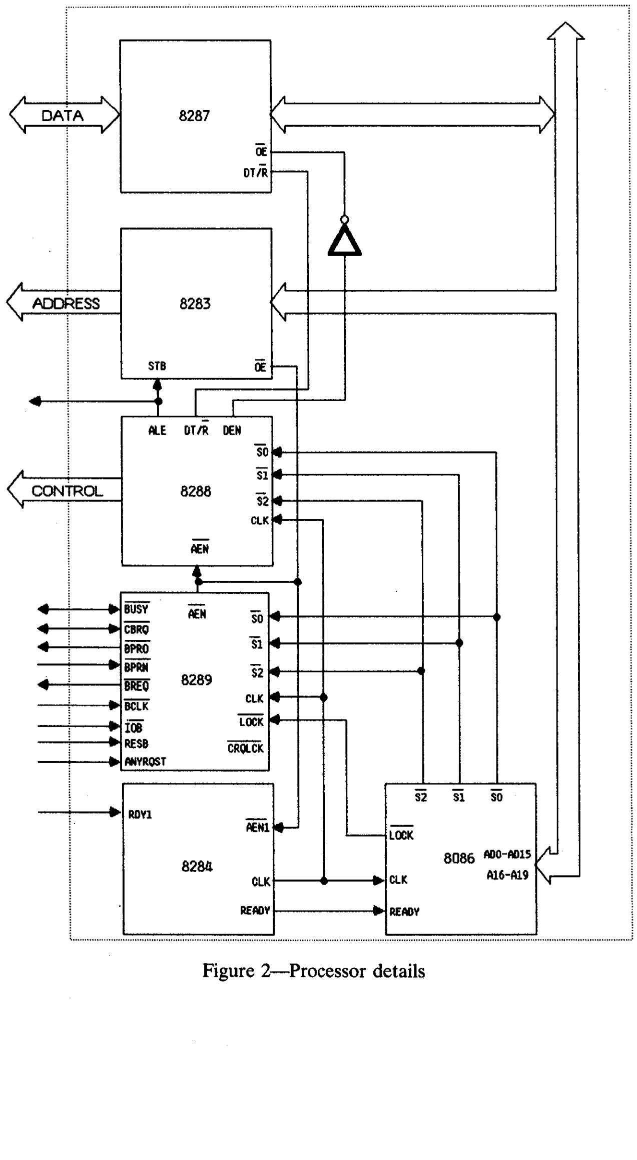


Figure 2—Processor details

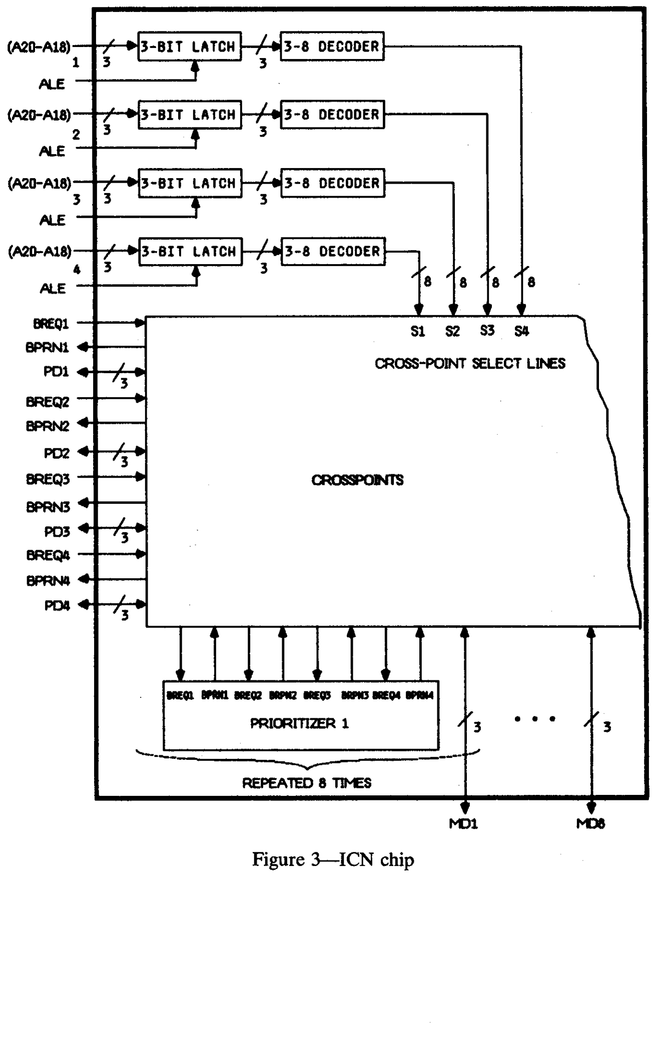


Figure 3—ICN chip

mode ensures the BREQ will not be activated until after the ALE (address latch enable) line is activated. Once the crosspoints have been set, the BREQ and BPRN (bus priority in) lines connect to the prioritizing logic for the memory requested. At the same time the BUSY and data lines (one set of bidirectional lines PD1, . . . , PD4) are also connected to those of the memory selected. A parallel priority resolving technique is used to decide which processor should have a memory in the event of a simultaneous request. The BREQ and BPRN lines of each arbiter requesting a given memory simultaneously are connected via the crossbar to the appropriate prioritizing logic for that memory. When the BREQ lines are activated, the prioritizer will return a BPRN active signal to the arbiter with the highest priority. Its associated processor will then take control of the memory as soon as it is no longer busy (BUSY line inactive). BUSY is an active low OR-tied signal that is connected via the crossbar to every arbiter in the system. Every memory has a BUSY line associated with it. When the BUSY line becomes inactive, the arbiter with priority takes control of the memory by activating BUSY to prevent any other arbiter from taking the memory. The arbiter maintains its BREQ active throughout the time it is connected to a memory.

Memory Surrender Conditions

It is possible for a processor to switch between memory modules from one bus cycle to the next. If we allow an arbiter to keep a memory between bus cycles, the possibility exists that a processor could access a busy memory in a subsequent bus cycle and cause a conflict. This problem could arise if the location of the present bus cycle address is in a different memory from the memory addressed during the previous bus cycle. The arbiter would assume it has control of the memory and instruct its attached bus controller and 8086 to proceed with the bus cycle as usual. A conflict would arise if this new memory is already busy. We have therefore configured our system so that an arbiter gives up the memory under its control after completing a bus cycle regardless of whether another processor is requesting that memory or not (packet switched). This is done by strapping ANYRQST (any request) high and CBRQ (common bus request) low. CBRQ is an open collector signal of the arbiter which can function either as an output or an input, depending on whether its associated arbiter has control of a memory. As an output it is sent by a lower priority arbiter to request a memory from a higher priority arbiter. As an input it instructs the arbiter presently controlling a memory that a lower priority arbiter would like the memory. ANYRQST is a signal of the arbiter that when strapped high signifies that the arbiter should release the memory after the end of the current bus cycle. A provision could be made to control CRQLCK (common request lock) via an I/O port from the processor. Activating CRQLCK prevents the 8289 from releasing the memory to any processor having a lower priority. In this way a processor could retain a memory as long as no other processor with higher priority requested it (circuit switched). One would have to make sure, when using such a provision, that while CRQLCK is activated all instructions and data reside in one memory.

ANALYTICAL MODEL

The behavior of the multiprocessor system described above can be approximated by a stochastic process, given the following assumptions about its operation. At the beginning of the bus cycle a processor selects a memory module at random with probability $1/M$ (there are M memory modules) and makes a request to access that module with probability r (≤ 1). If more than one processor requests the same memory module, the arbitration logic will choose the processor with higher priority. The other processors will continue to request the memory module until they are allowed access. The time it takes to perform this arbitration, i.e., to set up the processor-crossbar interface logic, is two bus cycles. The connection time between the processors and the memory modules will last for C bus cycles. A processor has at most one request waiting to be serviced at any time. During operation the behavior of each of the processors is considered to be independent but statistically identical. The memory access priority assigned to each of the processors by the prioritizing logic in the ICNs is as follows: the first processor has the highest priority in the first two memories, the second highest priority in the second two memories, the third highest priority in the third two memories, and the fourth highest priority in the fourth two memories; the second processor has the first highest priority in the second two memories, the second highest priority in the third two memories, the third highest priority in the fourth two memories, and the fourth highest priority in the first two memories; a similar pattern is repeated for the third and fourth processors.

The behavior of the multiprocessor system, under the operation assumptions stated above, can be described by using a discrete-time Markov chain. However, such a chain has an unmanageably large state space.[7] To avoid this, an approximate model can be used. In this study, we will use the equivalent rate model.[8] This model assumes that a memory module that receives more than one request selects any one of the requesting processors equiprobably, i.e., all the processors are modeled as having the same priority. Furthermore, the model assumes that the steady-state flow of the processors to the memory modules is equal to the flow of the processors from the memory modules: the equivalent rate model is an extension of the steady-state flow model.[9] The derivation of the model proceeds as follows: the rate of requesting a multiple bus cycle connection, r, is transformed to the equivalent rate of requesting a single bus cycle connection, r_{eq}. The quantity r_{eq} represents the connection time as a fraction of the total average processor cycle which contains both the think time and the connection time. The think time of a processor, T, is the time elapsed between releasing a memory module and making the next request for memory connection. Hence, r_{eq} is expressed as follows:

$$r_{eq} = \frac{C'}{T + C'}$$

where $T = 1/r - 1$ and $C' = C + 2$. The term C' includes the two bus cycles that are needed to set up the processor-crossbar interface logic. In our case where the crossbar is operating in packet switched mode, $C = 4$ bus cycles. A measure of the effectiveness of the crossbar is its bandwidth, BW, which can be expressed as follows:

$$BW = N\,U_p\,r_{eq}$$

where U_p is the processor utilization, i.e., the probability that a processor is thinking or accessing a memory module. Hence, $N\,U_p$ is the expected number of processors thinking or accessing; and $N\,U_p\,r_{eq}$ is the expected number of processors accessing, i.e., the crossbar bandwidth. Furthermore, it can be shown[9] that

$$BW = M\left[1-\left(1-\frac{U_p r_{eq}}{M}\right)^N \left(1-\frac{1}{M}\left(1-\left(1-\frac{1-U_p}{M}\right)^N\right)\right)^M\right].$$

The above two equations for BW can be solved by iterating on U_p, with $N = 4$ and $M = 8$ (two or three iterations are usually sufficient). The resulting value for BW is the memory bandwidth of the system, assuming that the connection time is C' bus cycles rather than C bus cycles. Therefore, the true memory bandwidth of the multiprocessor system described above, BW_{true}, is given by

$$BW_{true} = \frac{C}{C'}BW.$$

To test the validity of the approximate model described above, the results of the model are compared to those obtained by simulation. A SIMSCRIPT II.5 simulation program has been used to simulate the multiprocessor system. Table I shows the simulation and the model results for the memory bandwidth, given different values of the request rate, r. In addition, the simulated memory bandwidth is compared to the ideal memory bandwidth of the system, BW_{ideal}, which is defined as follows:

$$BW_{ideal} = 4 \times \frac{C}{T+C}.$$

In other words, BW_{ideal} is the memory bandwidth obtained assuming that each of the four processors operates independently without interference during memory accesses. Hence, BW_{ideal} can be thought of as the maximum potential memory bandwidth for the multiprocessor system. The percentage difference between the simulated BW and BW_{ideal} is also shown in Table I. $\%Diff$ is defined as follows:

$$\%Diff = \frac{BW_{ideal} - Simulated\ BW}{BW_{ideal}} \times 100$$

The simulation used priorities determined by the prioritizer logic in the ICN's, nevertheless the model, in which all priorities are equal, produces similar results to simulation. This arises because the queueing discipline for memory modules does not affect BW. The results presented in Table I show, among other things, that the performance of the multiprocessor system is degraded in the region where r is small. Since the memory conflicts are minimal in this case this degradation is almost entirely the result of the setup time. By comparing with cases where setup is fixed at zero, it can be shown that the effects of the memory conflicts become more critical as r increases. However, even in the case $r = 1$ the degradation due to interference never rises above 12%, i.e., about 33% of the degradation is due solely to setup (see last entry in Table I). Finally, measurements performed on a number of 8086 systems indicate that $r = 0.4$ is a frequent operating point.

TABLE I—Comparisons between the simulation and the model results

r	Simulated BW	BW_{true}	$\%Diff$
0.1	1.04230	1.03091	15.31
0.2	1.49357	1.47896	25.32
0.3	1.72489	1.71427	31.72
0.4	1.86074	1.85618	36.04
0.5	1.95659	1.95026	38.86
0.6	2.02529	2.01692	40.39
0.7	2.07624	2.06653	42.53
0.8	2.12421	2.10483	·43.58
0.9	2.14873	2.13527	44.79
1.0	2.17244	2.16003	45.69

CONCLUSION

In this paper we have presented a technique for constructing MIMD multiprocessors using a 3 bit slice component, the ICN chip. A performance model was presented that showed close agreement with simulation, and thus would make a useful design tool for estimating crossbar performance. The main measure of crossbar efficiency, BW, was shown to be dependent mainly on the setup time in the case of a four-processor/eight-memory system. In the case where processors have local memory accessed through a resident bus the values for r (the request rate to the shared memory) are likely to be very small if the hit rate to local memory is high. In such systems a concentrator $(N > M)$ version of the ICN chip would be more appropriate.

ACKNOWLEDGMENT

The work described in this paper was supported in part by Air Force Contract No. F49620-82-C-0089.

REFERENCES

1. Makrucki, B. A., and T. N. Mudge. "VLSI Design of a Crossbar Switch." SEL Report No. 149, Department of Electrical and Computer Engineering, University of Michigan, January 1981.

2. McFarling, S., J. L. Turney, and T. N. Mudge. "VLSI Crossbar Design Version Two." CRL Report CRL-TR-8-1982, Department of Electrical and Computer Engineering, University of Michigan, February 1982.
3. Wu, C. W. C. "Interconnection Networks," *Computer*, 14 (1981), 12, 8–9.
4. Kruskal, C. P., and M. Snir. "The Importance of Being Square." *Proceedings of the 11th Annual International Symposium on Computer Architecture,* IEEE Computer Society, Ann Arbor, MI, 1984, pp. 91–98.
5. Franklin, M. A. "VLSI Performance Comparison of Banyan and Crossbar Connection Networks." *Proceedings of Workshop on Interconnection Networks,* 1980, pp. 20–28.
6. *iAPX 86,88 User's Manual.* Santa Clara: Intel Corporation, July 1981, pp. A.111–A.134.
7. Skinner, C. E., and J. R. Asher. "Effects of Storage Contention on System Performance." *IBM Systems Journal, 8* (1969), 4, pp. 319–333.
8. Mudge, T. N., and H. B. Al-Sadoun. "Memory Interference Models with Variable Interconnection Time." *IEEE Transactions on Computers, C-33* (1984), pp. 1033.
9. Yen, D. W. L., J. H. Patel, and E. S. Davidson. "Memory Interference in Synchronous Multiprocessor Systems." *IEEE Transactions on Computers, C-31* (1982), pp. 1116–1121.

The PASM prototype interconnection network design

by NATHANIEL J. DAVIS IV and HOWARD JAY SIEGEL
Purdue University
West Lafayette, Indiana

ABSTRACT

Computer system design costs traditionally have been dominated by the development and production costs of hardware. The availability of large-scale integrated circuitry has dramatically cut the cost of system hardware to the point where new architectural designs can be built and tested at a comparatively low cost. The PASM prototype currently under development will be used to validate the design principles and operations of a large-scale, dynamically reconfigurable parallel processing system. The interconnection network of the PASM prototype, an implementation of a circuit-switched extra stage cube topology, is described in this paper. Network design tradeoffs and implementation options are discussed and related to the performance, development cost, and production cost of the network and the overall prototype system.

INTRODUCTION

During the past decade, the cost of designing and building a new computer system has changed radically. With the advent of very large scale integrated circuit technology, relatively inexpensive hardware systems and subsystems are readily available. The modern computer system architect now can design, build, and validate new designs that would have at best been "paper designs" only a few years ago. Research and validation such as this is being conducted, for example, at the University of Texas at Austin (TRAC[1]), at New York University (Ultracomputer[2]), and at Purdue University (PASM[3]).

PASM (partitionable SIMD/MIMD computer system) is a large-scale, dynamically reconfigurable multimicroprocessor design that can be partitioned to operate as several independent SIMD/MIMD machines of various sizes. In its full implementation, PASM will incorporate more than a thousand complex processing elements. The prototype currently under construction will be a scaled-down, 16-processing-element version of the full PASM system. The prototype will be used to validate the overall system design concepts and, once it is operational, will be used as a research tool in the investigation of the uses of parallel processing in tasks such as speech and image understanding. The prototype design attempts to incorporate the flexibility needed for studying large-scale SIMD (synchronous) and MIMD (asynchronous) parallelism,[4] while keeping the system developmental time and costs reasonable. The overall prototype design process has been geared to meeting this objective rather than producing the fastest, the biggest, or the most technologically advanced system.

A block diagram of the major system components of PASM is shown in Figure 1. The PASM prototype design is based on the use of the Motorola MC68000 16-bit microprocessor chip family as the basic computational device in the system. Only off-the-shelf components are being used in the prototype to minimize development time and development and construction costs. It is, however, anticipated that the implementation of the full PASM system will capitalize on the use of custom-designed VLSI components throughout the system.

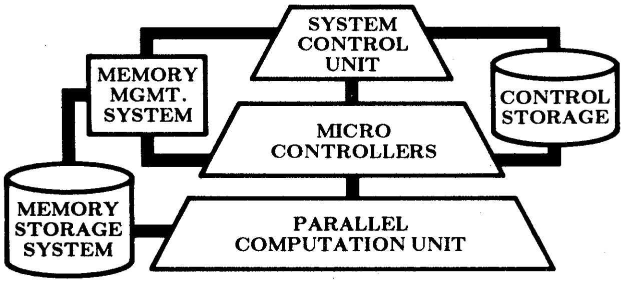


Figure 1—Block diagram of the PASM system components

This paper describes the detailed design of the interconnection network that is to be used within the parallel computation unit. More information about the PASM computer system and list of selected references for further reading about PASM appear in Reference 5.

INTERCONNECTION NETWORK DESIGN SPECIFICATIONS

The parallel computation unit (PCU), shown in Figure 2, contains $N = 2^n$ processors ($N = 1024$ for the full system, 16 for the prototype), N memory modules, and an interconnection network. The processors are microprocessors that perform the actual SIMD and MIMD computations. The memory modules are used by the processors to store data in the SIMD mode and data and instructions in the MIMD mode. A processor and a memory module are paired together to form a processing element (PE). The function of the interconnection network within the PCU is to provide a means of performing inter-PE data communication. In an SIMD environment, the network is used to perform data permutations. In this environment, data transfers through the network will be controlled in a manner that will eliminate network congestion. In contrast, MIMD operations can be characterized by their inherent randomness—in source–destination pairings, message lengths, and loading probabilities.

The prototype interconnection network design must be able to support SIMD and MIMD operations, perhaps simultane-

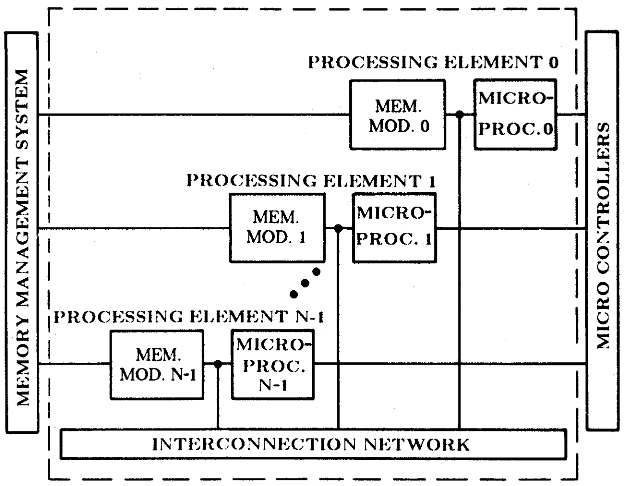


Figure 2—Parallel computation unit block diagram

ously if the system is partitioned into independent subsystems. Additionally, the design must be theoretically scalable to a full PASM interconnection network. To design the network in a reasonable amount of time and at a reasonable cost, the following guidelines were formulated:

1. The network should operate at transmission rates high enough to prevent the network from acting as a bottleneck to the computer system in general.
2. The network must be fault tolerant.
3. Design complexity should be minimized by maximizing the use of readily available electronic parts and assembly techniques.
4. The network must be capable of being dynamically partitioned into smaller, independent subnetworks.
5. The network should be capable of supporting message transfers under direct memory accessing (DMA) control.
6. Interfacing the network to the PEs is through the use of the Motorola MC68230 parallel interface and timer chip.

NETWORK TOPOLOGY

Two basic types of networks have been considered for use in the PASM prototype: the generalized cube network and the augmented data manipulator (ADM) network.[6] The generalized cube network is representative of all of the multistage cube networks,[6] such as the SW-banyan (S = F = 2),[7] the baseline,[8] and the omega.[9] This type of topology is used in the STARAN machine[10] and in the proposed Ultracomputer.[2] Similarly, the ADM network is representative of the class of multistage PM2I networks,[6] such as the data manipulator,[11] the gamma,[12] and the inverse ADM.[13] Both the cube and the ADM networks consist of n stages of switches, are partitionable into independent subnetworks, can be controlled in a distributed manner, and can be used to link any source–destination pair.

The principal advantage of using the ADM network in the prototype will be its robustness when operating in a faulty or congested network environment. The majority of the ADM network's source–destination pairs have multiple, redundant communications paths, any of which can be used to complete the desired communication. The cube network is not fault tolerant; any single-point failure within the network will prevent some source–destination PE pairs from communicating. The lack of fault tolerance in the cube network is perhaps its greatest drawback.

Fault tolerance can be introduced into the cube network through the use of one or more extra stages of switches.[14–18] In a survey of a variety of fault-tolerant network approaches,[15] it was shown that the extra stage cube (ESC) network gives a significant improvement in fault tolerance at a comparatively low cost in terms of additional hardware requirements and operational complexity.[14]

The cube network has been selected for use in the PASM prototype. It provides the necessary permutation abilities needed in most SIMD operations[9,19] and reasonable responsiveness can be obtained in MIMD operations.[20] Additionally, in SSI–MSI implementations, the cube network is significantly less complex than the ADM network.[21] To provide the fault tolerance needed within the interconnection network, the cube network will be implemented as the ESC network.

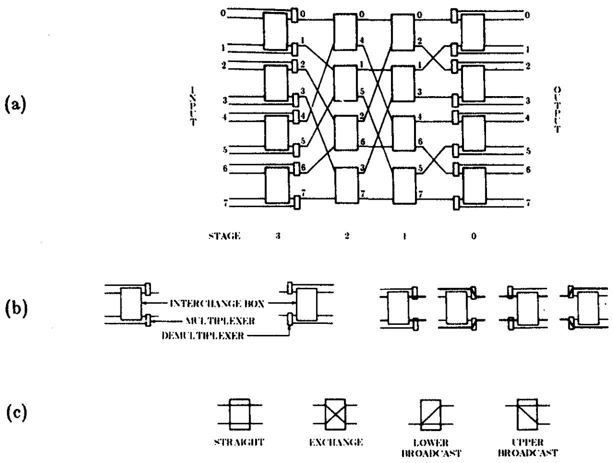


Figure 3—(a) Extra Stage Cube network for N = 8; (b) implementation of stage 0 and stage n; (c) allowable box settings

The ESC network has $n + 1$ stages (labeled 0 to n), where each stage consists of a set of $N/2$ interchange boxes. This is shown in Figure 3 for $N = 8$. Each interchange box is a 2-by-2 crossbar switching device that can be set to one of the four legitimate connection states shown. The network implementation is unidirectional, with PE j sending data to the network via input j and receiving data from the network via output j. Interchange boxes in stage i pair input–output lines that differ only in the i-th bit position. The same labeling is used for both the input and the output lines connected to an interchange box. The multiplexers and the demultiplexers in Figure 3 allow stages n and 0 to be bypassed (deactivated). By activating one or both of these two stages, any single fault in the network can be avoided. Parallel, redundant lines provide fault-tolerant connections between the input and output stages of the network and the PEs.[16]

NETWORK DESIGN

Network design alternatives, such as switching methodology (packet or circuit switching), message-routing schemes, network control, and operation in a faulty environment, have been considered in detail. The next section is an overview of these aspects that relates them to implementation of the network design.

Switching Methodology

The choice of implementing packet or circuit switching can greatly affect the performance of the network and, in turn, the system as a whole. In intercomputer networks, the classic tradeoff between circuit and packet switching is one of message length. The differences in the two switching methods are not as clear-cut in intracomputer networks found in parallel processing systems. System characteristics such as the operational mode (SIMD or MIMD), the architecture supporting the network, and the anticipated message format and system loading greatly influence the selection of the switching method.

Systems similar to PASM, which uses the network for inter-PE data transfers (and not for instruction fetching), can be characterized by lightly loaded, low message conflict networks (a result of high local PE processing efficiency) and potentially long message lengths (when compared with packet lengths in a packet switched network). In this case, circuit-switched networks can give acceptable performances using circuitry that is much less complex than that found in packet-switched networks. For example, message queues and their control logic are not needed in the interchange boxes. With circuit-switched networks, once the path connecting the source–destination PE pair is established, the switching elements contribute only minimal gate delays to the transmission time. The data can be transferred at an extremely fast rate—constrained only by the transmission rate of the PEs and the propagation delay of the path through the network. In packet switching, if the message exceeds the packet length, multiple packets must be sent through the network. The PEs must absorb the added overhead of message packetization, at the source, and packet recombination, at the destination. Each packet must perform its own routing–path establishment independently. These delays, coupled with the queuing delays at each interchange box, can increase the overall message transmission significantly.

In the PASM prototype, DMA operations would be difficult to support with a packet-switched network. The operation of the PE's MC68450 DMA controller chip is such that it expects to "see" a direct path to the destination. This would force the size of the DMA data transfers to be, at most, the network packet size. Packet switching would therefore lead to inefficient system performance because DMA operations typically are used for the transmission of large data block transfers. The inefficiency of packet-switched DMA operations could be improved by using an intelligent PE–network interface processor. Its use would, however, significantly add to the complexity of the network design.

Based on the projected mix of SIMD and MIMD PE processes, the anticipated DMA operations, and the projected network loading owing to inter-PE communication, a circuit-switched network was determined to be the best at meeting specified performance and design criteria. The implementation of designs for the prototype circuit-switched ESC interconnection network are discussed below.

Distributed Routing Control

Two different approaches can govern the way connections are made in a network: centralized control or distributed control. Centralized control is a global network control element that mediates between message requests and establishes the desired network connections. In contrast, distributed control removes this serial bottleneck by allowing the individual interchange boxes to establish their own connections based on the use of routing tags associated with each message. Two functionally equivalent forms of routing tags are the destination address tag[11] and the exclusive-or (XOR) tag.[7] In either case, each interchange box examines the routing tags of the messages at its input ports and makes the switching connections accordingly.

For example, let S be the source PE, D be the destination PE, and T be the routing tag for the destination address tag, T = D. The XOR tag is S⊕D. Without alteration, the XOR tag can be used by the destination PE for a return message routing tag and can be used to compute the source PE number (S = T⊕D). The destination tag requires no computation (i.e., the XOR function) and, by comparing it with the destination PE number, allows verification of the message routing. The source PE number must be explicitly included in the message because it cannot be derived from the destination address tag. Both routing schemes require the same number of bits and interchange box decoding complexity.

Because the destination address method is simpler to encode, however, it was chosen for implementation. The message will have the following format: The first word will be the routing tag itself. The second word will contain the source PE address. The remaining words in the message will be the data that are to be transferred. The routing tag is composed of two parts: the destination address tag and the broadcast mask. Interchange boxes in stage *i* will examine bits *i* of both the destination address tag and the broadcast mask. If the mask bit is "1," then a broadcast connection to both outputs is to be performed. If the mask bit is "0," then the destination address tag bit specifies the message's desired connection pattern—a "0" indicates connection to the upper output and a "1" indicates connection to the lower output.

Network Control Signals

The operation of the network will be controlled through the use of two types of asynchronous control protocols. The first is a message-request–grant protocol that is used in establishing a path that connects the source and destination PEs. The message request is the combination of the routing tag and a message-request signal, REQ. The message request will propagate through the network toward the destination PE. At each interchange box encountered in the network, the desired path (straight, exchange, or broadcast) is established, if possible.

If at some interchange box the desired connection cannot be made because the request conflicts with another message—that is, messages at both inputs to the box request the same output port—a conflict-handling algorithm must be invoked to resolve the conflict. The two most common conflict handling algorithms are the drop and the hold algorithms. In the drop algorithm, a conflict at an interchange box will cause the message request to be dropped from the network and be re-initiated by the source PE. The path being held by the request will be relinquished. The hold algorithm, in contrast, permits the message request to be "held" at the point of conflict until the conflicting message completes its transmission. The portion of the path that has already been established is held. The usefulness of these two algorithms is a subject of current research.[22] The prototype is being implemented so that both algorithms can be exercised under various operating environments to study their relative effectiveness.

The second type of asynchronous control used in the network is the handshaking signals generated by the parallel I/O ports that interface the PEs to the network. Data is transferred between the two ports using an asynchronous "data available–data received" protocol. The actual data transmission will not begin until the message-grant signal and the data-received signal for the routing tag are returned to the source PE.

Interchange Box Design

The basic interchange box used within the prototype design will be a 2-by-2 crossbar switch. As shown in Figure 4, the interchange box is divided into two sections: the crossbar switch itself and the control unit. The data path throughout the network is 22 bits wide. Sixteen bits are allocated for the transfer of the data word and two additional bits are used for parity check bits. The remaining 4 bits are used for protocol and handshaking between the source and destination PEs. The 3 bits (of the 22) that go to the control unit in Figure 4 are the *i*-th bits of destination address tag and the broadcast mask, and the REQ signal. The crossbar switch will be implemented using tri-state buffer ICs. Four control lines are used to selectively activate the sets of tri-states to perform the desired switching connection. Conflicting connection requests in an interchange box are handled by the control unit.

To preclude the possibility of conflicts occurring in the switch, the control unit must know the status of both inputs. The status of each input can be determined by examining the four control lines ($\overline{I0}$ and $\overline{I1}$ for the upper input, $\overline{I2}$ and $\overline{I3}$ for the lower input). As a result, 5 bits of information must be examined prior to setting up a path in the switch: the routing and broadcast mask bits, the message request bit, and the pair of control lines associated with the other switch input. Because messages can be generated at random times by the source PEs, the control unit must be operated as a synchronous circuit where the inputs are alternately checked for message requests. If not, it is possible that an invalid switch setting, resulting in a network error, could occur.

The main portion of the control unit is a PROM that contains two independent sections of controlling logic—one for each input. The 5 bits of control information for each input are gated into a register and, in turn, applied to the PROM. The PROM outputs are the four control lines that drive the tri-state buffers. The state of the control line pairs is held by another set of registers at the output of the PROM. Once set, a control line pair will not be reset until the associated message completes its transmission, at which time the source PE will reset the REQ line. The control unit in each interchange box in the transmission path, upon seeing the REQ reset, will reset the control line pair, disabling the associated tri-state buffers and releasing the path.

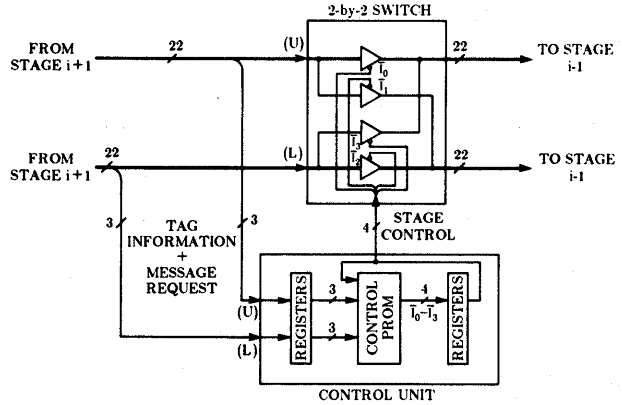
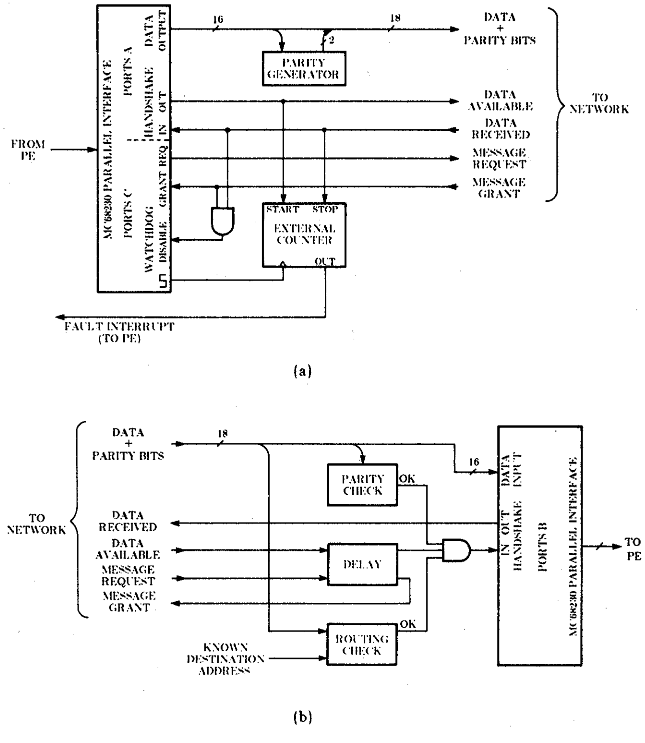


Figure 4—Interchange box implementation

PE-to-Network Interface

The PEs access the interconnection network through the PE–network interface. The interface consists of two distinct subsections, shown in Figure 5. Each MC68230 parallel interface IC is organized into three ports. Ports A and B are general purpose 8-bit I/O ports. Two MC68230s are used in parallel so that a full 16-bit data word can be written to or read from the I/O port in one PE operation—one byte going to and from each IC. Ports A are used as the PE's output port (to the network); ports B are used as the PE's input port (from the network). The third port on each MC68230 (Port C) is used for general purpose, bitwise I/O lines in support of the message-request–grant control protocol. It also provides control lines to activate or deactivate stage 0 and stage *n* (not shown in Figure 5).

An internal, programmable timer also can be accessed via Port C of the MC68230 chip. This timer is used in a "watchdog" mode, and its programming will determine whether the



(a)

(b)

Figure 5—PE- network interface. (a) PE-to-network subsection; (b) network-to-PE subsection.

hold or the drop conflict resolution algorithm is used in network operations. At the start of a message transfer, the watchdog timer is set to a predetermined (software programmable) value and begins to count down toward zero. Receipt of the combination of the message-grant signal and the data-received signal indicates that the routing tag has been received and verified by the destination PE and that the path is established and will be used to disable the watchdog timer. If the watchdog times out before being disabled, the PE will assume the message request was blocked in the network. It will then reset the message request signal (releasing the partially established path) and re-initiate the message later. If timeouts that occur over many attempted re-initiations indicate a network fault, the network fault-handling procedures will be invoked. The duration of the watchdog count determines which conflict algorithm is in use. If the watchdog is set initially to an extremely large (effectively infinite) value, it will not time out when blockages occur in the network and, as a result, the path will not be relinquished—this is the hold algorithm. On the other hand, if the initial value is set to the time required to set up a path without conflicts, then any conflicts encountered in the network will delay the grant- and data-received signals enough to allow the watchdog to time out, causing the path to be dropped—an approximation of the drop algorithm.

When a data word arrives at the network-to-PE subsection, it is applied to the parity-checking logic. If the word is the first word of a message, then the routing tag portion is compared with the known destination address to confirm the message routing. If both the parity check and the routing verification are correct, the word is then gated into ports B of the MC68230s, and the data-received handshake signal is returned to the source PE. If an error is detected, the faulty data word is *not* gated into the destination PE and, as a result, no data-received handshake signal is generated.

The external counter in Figure 5a is used to track the transmission delay of each data word. Use of the watchdog timer for this would incur significant system overhead in resetting the watchdog after each word is transmitted. The delay is the elapsed time between the assertion of the data-available signal and the receipt of the data-received signal (both at the source PE). The counter is driven by a programmable frequency square-wave signal that is available on port C of the MC68230s. If the data-received signal does not disable the counter before its maximum count value is reached, an error is assumed to have occurred. An error interrupt is generated that causes the PE to attempt to retransmit the data word that is assumed faulty or jump to the network fault-handling routines, depending on the perceived severity of the fault.

Operation in a Faulty Environment

Once a fault has been detected in the network, the exact location of the fault must be determined and broadcast to all of the PEs. The exact location of the fault can be determined explicitly through the use of off-line network test messages.[23–25] Based on the location of the fault, the extra stage of switching elements can be activated to operate in conjunction with, or in place of, the stage 0 interchange boxes to guarantee at least one fault-free path for every source–destination pairing. When a PE initiates a message in the presence of a network fault, it must compare the known fault location with its message path. If the fault location corresponds to a needed interchange box or link, the PE can invoke the alternative, fault-free message routing path by modifying the routing tag.[6] Because halting all PEs within the affected subnetwork to perform fault location routines is inefficient, PEs that detect potential faults will try to retransmit the assumed faulty word or reestablish the assumed incorrectly routed path before reporting the fault to the system control unit. The number of retransmission attempts is controlled by software to permit the maximum flexibility in use of the system.

Construction Notes

The prototype ESC network will require a total of 40 2-by-2 interchange boxes. Counting the interchange boxes and the PE-to-network interface circuitry, the interconnection network will use 1200 integrated circuits. Using the worst-case delay specifications for the individual ICs, the prototype network will be able to establish a path through the network in 1200 nanoseconds with SIMD operation or lightly loaded MIMD operation. Once a path has been established, data words can be transferred through the network at a rate of one 16-bit word every 400 ns. Because it takes at least two processor memory cycles (800 ns for a 10 Mhz system clock) to transfer a single word from a PE to its network interface port, the network will not act as a bottleneck in the processing ability of the overall system. An 8-bit data path through the network was not implemented because the overhead required to make it compatible with the 16-bit ports would cause the transfer time to exceed 800 ns.

The design of the prototype network is scalable to larger systems. The size of the network will, however, limit the system size that can reasonably be supported. It is expected that network designs for systems substantially larger than the current prototype will rely heavily on VLSI implementations of interchange boxes, most likely with 4 or 8 I/O lines per box. The current prototype will allow verification of communication techniques needed for these systems.

SUMMARY

The PASM prototype interconnection network provides a means for communication among the system's processing elements. The availability of SSI and MSI ICs has been exploited to reduce development time and costs. The circuit-switched design uses a 16-bit-wide data path with distributed routing control that is fault tolerant through the implementation of the ESC and can operate using either the drop- or the hold-conflict resolution algorithm. The prototype network can support DMA data transfers between PEs as well as dynamic partitioning into independent subnetworks. The worst-case data transmission rate of the network exceeds the rate at which the PEs can generate data so as not to be a bottleneck in the system. The prototype is scalable to larger system sizes. When it is integrated with the complete PASM prototype system it will allow the verification of the fundamental architectural concepts of the full PASM system.

ACKNOWLEDGMENT

This research was supported by the Rome Air Development Center, under contract F30602-83-K0119. Discussions with Dave Meyer, Jim Kuehn, and Tom Schwederski and the work of Bill Hsu have been very helpful and are greatly appreciated.

REFERENCES

1. Sejnowski, M. C., E. T. Upchurch, R. N. Kapur, D.P.S. Charlu, and G. J. Lipovski. "An Overview of the Texas Reconfigurable Array Computer." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 49), 1980, pp. 631–641.
2. Gottlieb, A., R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir. "The NYU Ultracomputer—Designing an MIMD Shared Memory Parallel Computer." *IEEE Transactions on Computers,* C-32 (1983), pp. 175–189.
3. Siegel, H. J., L. J. Siegel, F. C. Kemmerer, P. T. Mueller, H. E. Smalley, Jr., and S. D. Smith. "PASM: A Partionable SIMD/MIMD System for Image Processing and Pattern Recognition." *IEEE Transactions on Computers,* C-30 (1981), pp. 934–947.
4. Flynn, M. J. "Very High-Speed Computing Systems." *Proceedings IEEE,* 54 (1966), pp. 1901–1909.
5. Siegel, H. J., T. Schwederski, N. J. Davis IV, and J. T. Kuehn. "PASM: A Reconfigurable Parallel System for Image Processing." *Workshop on Algorithm-Guided Parallel Architectures for Automatic Target Recognition.* 1984, in press; reprinted in *ACM SIGARCH Computer Architecture News,* 12 (1984), pp. 7–19.
6. Siegel, H. J. *Interconnection Networks for Large Scale Parallel Processing: Theory and Case Studies.* Lexington, Mass.: Lexington Books, 1984.
7. Goke, L. R., and G. J. Lipovski. "Banyan Networks for Partitioning Multiprocessor Systems." *1st Annual International Symposium on Computer Architecture.* 1973, pp. 21–28.
8. Wu, C., and T. Feng. "On a Class of Multistage Interconnection Networks." *IEEE Transactions on Computers,* C-29 (1980), pp. 694–702.
9. Lawrie, D. H. "Access and Alignment of Data in an Array Processor." *IEEE Transactions on Computers,* C-24 (1975), pp. 1145–1155.
10. Batcher, K. E. "The Flip Network in STARAN." *1976 International Conference on Parallel Processing.* 1976, pp. 65–71.
11. Feng, T-Y. "Data Manipulating Functions in Parallel Processors and Their Implementations." *IEEE Transactions on Computers,* C-23 (1974), pp. 309–318.
12. Parker, D. S., and C. S. Raghavendra. "The Gamma Network: A Multiprocessor Network with Redundant Paths." *9th Annual International Symposium on Computer Architecture.* 1982, pp. 73–80.
13. McMillen, R. J., and H. J. Siegel. "Routing Schemes for the Augmented Data Manipulator Network in an MIMD System." *IEEE Transactions on Computers,* C-31 (1982), pp. 1202–1214.
14. Adams, G. B. III, and H. J. Siegel. "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems." *IEEE Transactions on Computers,* C-31 (1982), pp. 443–454.
15. Adams, G. B. III, and H. J. Siegel. "A Survey of Fault-tolerant Multistage Networks and Comparison to the Extra Stage Cube." *17th Annual Hawaii International Conference on System Sciences.* 1984, pp. 268–277.
16. Adams, G. B. III,and H. J. Siegel. "Modifications to Improve the Fault Tolerance of the Extra Stage Cube Interconnection Network." *1984 International Conference on Parallel Processing.* 1984, pp. 169–173.
17. Padmanabhan, K., and D. H. Lawrie. "A Class of Redundant Path Multistage Interconnection Networks." *IEEE Transactions on Computers,* C-32 (1983), pp. 1099–1108.
18. Chin, C-Y., and K. Hwang. "Connection Principles for Multipath Packet Switching Networks." *11th Annual International Symposium on Computer Architecture.* 1984, pp. 99–107.
19. Pease, M. C. "The Indirect Binary n-Cube Microprocessor Array." *IEEE Transactions on Computers,* C-26 (1977), pp. 458–473.
20. McDonald, W. C., and J. M. Williams. "The Advanced Data Processing Test Bed." *Compsac,* 1978, pp. 346–351.
21. McMillen, R. J., and H. J. Siegel. "Evaluation of Cube and Data Manipulator Networks." *Journal of Parallel and Distributed Computing,* 2 (1985), in press.
22. Lee, M., and C-L. Wu. "Performance Analysis of Circuit Switching Baseline Interconnection Networks." *11th Annual International Symposium on Computer Architecture.* 1984, pp. 82–90.
23. Feng, T-Y., and C-L. Wu. "Fault-diagnosis for a Class of Multistage Interconnection Networks." *IEEE Transactions on Computers,* C-30 (1981), pp. 743–758.
24. Agrawal, D. P. "Testing and Fault Tolerance of Multistage Interconnection Networks." *Computer,* 15 (1982), pp. 41–53.
25. Lim, W. Y-P. "A Test Strategy for Packet Switching Networks." *1982 International Conference on Parallel Processing.* 1982, pp. 96–98.

Prototype of Star architecture—a status report

by CHUAN-LIN WU, MANJAI LEE, CHAI SUDTIKITPISAN, JAMSHID MOADDEB, GEOFFREY BROWN, WOEI LIN, NADER BAGHERZADEH, and DAVID VAUGHN

The University of Texas at Austin
Austin, Texas

ABSTRACT

A prototype machine of the Star parallel processing system is being built. The Star system employs reconfigurable interconnection networks and a distributed control mechanism for supercomputing. Progress on the design of interconnection networks, interface unit, operating system kernel and instruction set, and resource allocation is reported here to delineate concepts and status.

INTRODUCTION

Star, a distributed computing architecture, has been proposed for tightly-coupled multiprocessing.[1] The project is investigating the use of distributed control over parallel hardware and compilation techniques to fully exploit parallelism of algorithms for a suitable match between the architecture and algorithms. The system employs high bandwidth circuit switching interconnection networks and a distributed operating system (OS). Given a task, the OS can allocate not only processor resources, but also the necessary communication connections in a single step. The architecture is versatile enough to emulate various current architectures because many machine configurations can be formed. It is expected that applications can be processed at optimum speed by properly configuring resources to form relevant parallel structures.

The project is motivated by the problems associated with underlying computer architecture. In many cases, the communication structure of a program to be executed must correspond closely to the particular processor connection topologies assumed; otherwise, excessive overhead for task allocation and interprocessor communication is observed. Consequently, algorithms are executed at a sub-optimum speed. In addition, most computer architects think that interconnection networks of parallel architecture are strictly for the use of data communication among processors. The inherent impact of an interconnection network on system control is usually ignored. Consequently, in most architectures, the interconnection network is poorly designed.

This paper provides a hardware connection model, then reports the preliminary results in the development of an eight processor prototype. It includes results of the interconnection networks, the interface unit, the operating system kernel and instruction set, and resource allocation.

HARDWARE CONNECTION MODEL

The hardware model of Star, as illustrated in Figure 1, is composed of a collection of N processor nodes and a communication subnet called *Starnet.* Each processor node is connected to B baseline networks[2] through an interface unit. The baseline network is a multistage interconnection network with N inputs and N outputs. By properly setting the control of individual switching elements of the baseline network, circuit paths can be connected to allow interprocessor communication.

Each processor node can be assigned as a controller that can connect a set of free processor nodes in terms of a broadcasting tree formed in the Starnet. In response to a resource request of a task, a controller broadcasts a command to the free processor nodes, providing an allocation formula and the connection topology to be formed. Those allocated processor nodes concurrently calculate their connection destination and issue connection requests to form the requested topology in a single step.

The B baseline networks provide multiple path connections between two processor nodes. When B is set equal to $N/2$, where N is the number of I/O ports, the N processor nodes can be completely connected. However, due to practical engineering considerations and realistic bandwidth requirements, the number of baseline networks needed will probably not exceed four. A dual configuration consisting of two baseline networks is shown in Figure 2. An evaluation result[3] shows that two baseline networks are probably sufficient for most cases.

INTERCONNECTION NETWORKS

A prototype network consists of two baseline networks, a set of Interface Units (IUs), and a number of processor nodes. As shown in Figure 1, multiple baseline networks are used to provide versatile connection among processor nodes. Figure 3 shows an 8 × 8 baseline network. By manipulating the switching elements in the baseline network, we can form many different topologies. Because a baseline network has the

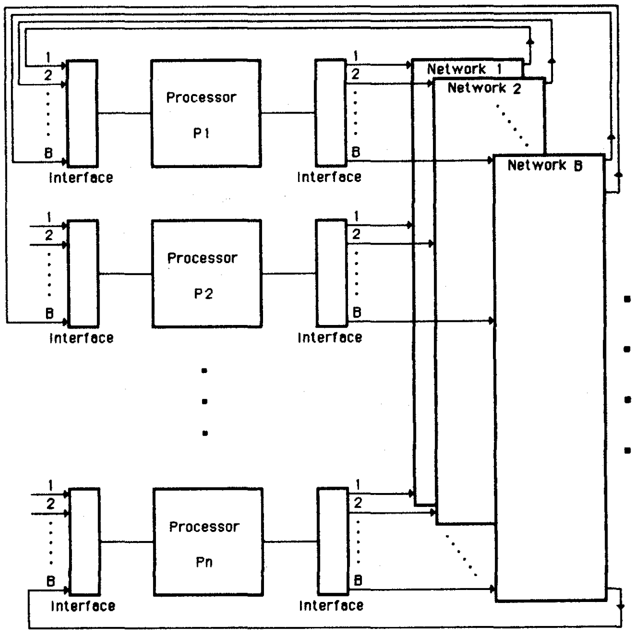


Figure 1—Star hardware model

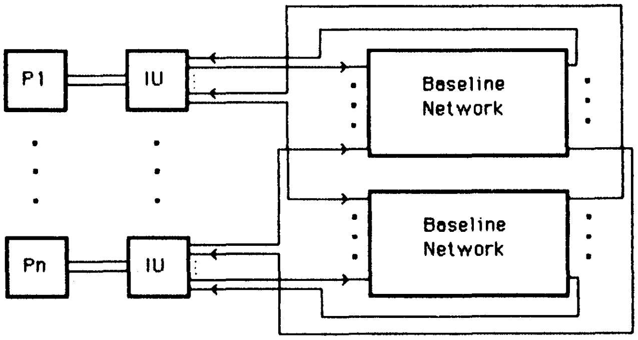


Figure 2—Starnet in dual configuration

blocking property, there might be a significant delay in some applications even though we carefully assign the processor nodes. The baseline network is composed of 2 × 2 switching elements, each having four ports (A, B, C, and D) as shown in Figure 4. The state of the switching element can be represented with four connections and their directions.

To fully utilize the bandwidth of the network, an asynchronous communication method is adopted so that once a path is set up, data transfer can be done at the maximum speed (which may be higher than the network clock speed). For setup of a path between a processor node and one or more other processor nodes, a synchronous operation is adopted to realize a reliable network.

The network is designed to support broadcasting and reconfiguration. By implementing the direction change and broadcast mechanism in the network, we can partition the network into several clusters and execute several different tasks concurrently. Each input to the interconnection network can be in one of five phases: setup, path configuration, data transfer, release, or idle. The state transition is diagrammed in Figure 5.

To establish a path, a processor node places the destination address and broadcasting information on the data lines and raises its Request (REQ) signal. Once the connection is made, the REQ signal is delivered to one or more receiving processor node which responds by raising its Acknowledge (ACK) signal. If more than one slave processor node is connected, the individual ACK signals are logically ANDed together and returned to the master processor node. The master processor node then lowers its REQ line and the slave processor nodes lower their ACK signals. This completes a setup of a path. The interconnection network allows several types of left-to-right connections: broadcasting (one to many), single (one to one), and merging (many to one).

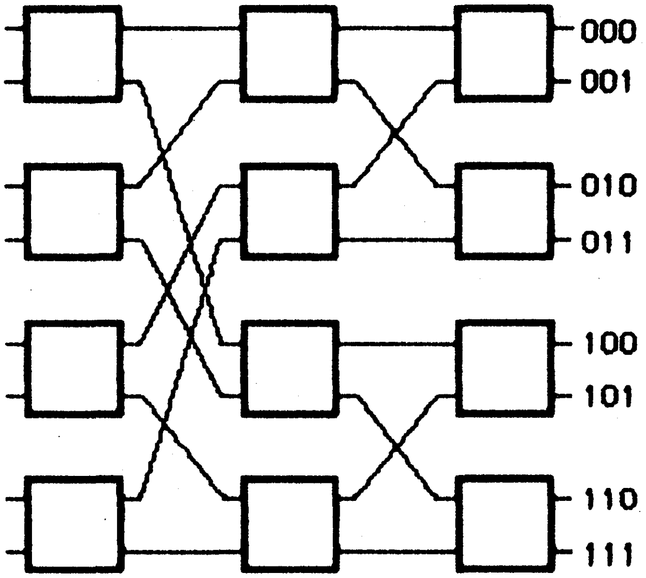


Figure 3—8 × 8 baseline network

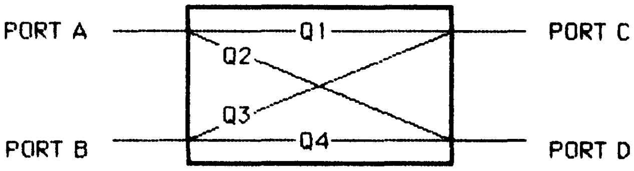


Figure 4—2 × 2 switching element

An established path can be shared between any of the processor nodes connected to it. The processor node that submitted the connection request is the master; processor nodes on the other side of the network are slaves. When the path is idle, the master processor node has the "talking right." When a slave processor node wants to communicate with other processor nodes, it raises its Path Request (PREQ) signal. All PREQ signals from the slaves are ORed and delivered to the master processor node which responds by sending a Path Grant (PGNT) signal to the switching element in the first stage of the network. This PGNT signal is delivered to only one requesting slave processor node. Conflicts are resolved at the switching elements. When the talking right is granted, the direction of the connection between the master processor node and the granted slave processor node is reversed. No other directions are changed. The new talker can send data through the path to any of the processor nodes attached to the path. After finishing the data transfer, the talker sends a PREL signal to the master processor node; this returns the direction of the communication to its initial state.

Once a path is set up, the master processor node can use it immediately. Other processor nodes can use it only after they get the talking right from the master processor node. Three signal lines are required for "handshaking" for each direction:

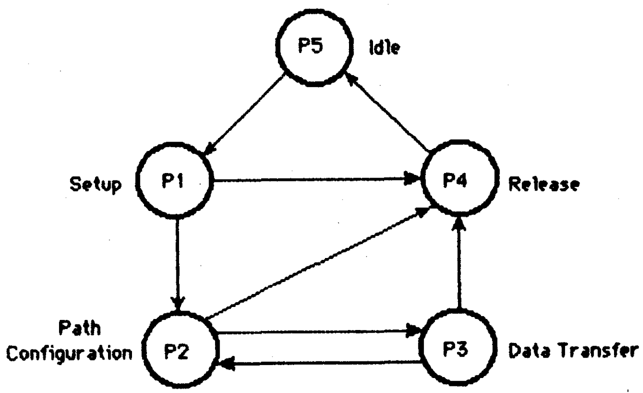


Figure 5—Phases of network operation

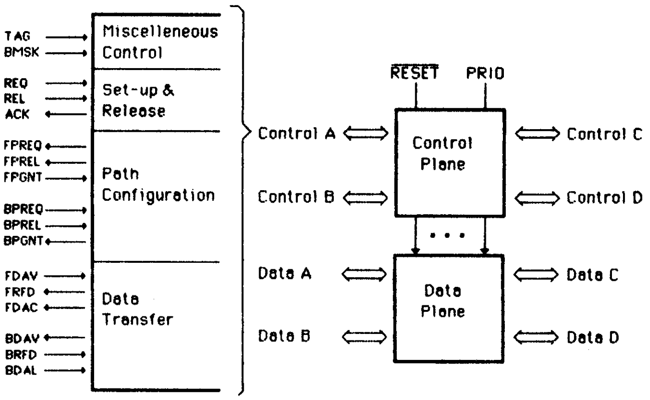


Figure 6—Signals of the switching element

Data Available (DAV), Request for Data (RFD), and Data Accepted (DAC). The names of the signals and operating principle are borrowed from the IEEE 488 standard, whose broadcasting capability has been already proven.

As can be seen in Figure 6, the switching element consists of two parts, the control plane and the data plane. The function of the control plane is to control the switching element when it is in one of the possible phases. The control plane must determine if a requested connection can be made. If there is no conflict, the control plane sends the correct control signals to the data plane. Once a connection is made, the control plane must determine which port has the talking right and must control the handshaking signals used in data transfer. As a building module of the network, the switching element receives signals from the stages on either side of it, relays the inputs to the correct outputs, and sends control signals to the data plane.

The network can be used either in Single Instruction stream—Multiple Data stream (SIMD)-mode, or in Multiple Instruction stream—Multiple Data stream (MIMD)-mode, depending on what kind of processor nodes it is attached to. In SIMD application, the communication paths are used mainly to send the instruction code to multiple processor nodes. One master processor node controls 2^k slave processor nodes for instruction execution and other control functions. The master processor node broadcasts instruction code to multiple slave processor nodes. Slave processor nodes respond with RFD and DAC to acknowledge that they have executed the instruction or other function requested by the master processor node. By using this acknowledgment scheme, variations in instruction execution time that might occur due to differences in the data can be tolerated. Logic equations of the switching element have been presented in detail.[4]

Bit	0/1	Description
b0	0 1	Ready state Not ready, CPU does not want any connection
b1	1 0	Interface has been configured in either master or slave mode. Interface is in idle mode. Writing 0 in this bit also reset the interface, e.g. PE is the master and want to release the connection.
b2	1 0	Master mode Slave Mode
b3	1 0	Talker mode Listener Mode
b5,b4	01 11 x0	CPU mode, read/write bypassing the FIFO CPU mode, read/write into the FIFO Using DMA service
b6	1	If b2=1(master mode), the interface would honor(grant) the request for the direction to be changed. If b2=0(slave mode), the interface is requesting the direction to be changed(wants to be talker). This signal should be left at 1 as long as the slave is a talker and changed back to 0 when it wants to return to listener mode.
	0	If b2=1, the interface would not honor the request for changing direction. If b2=0, the interface is in the normal condition(listener) or it is not requesting any direction change.

b7	b6	b5	b4	b3	b2	b1	b0

TABLE I—Interface unit control word

Bit	0/1	Description
b0	1 0	Connection request pending No connection request pending
b1	1 0	Port connected Port not connected
b2	1 0	Port in transmit mode Port in receive mode
b3	1 0	Direction change pending No direction change request
b4	1 0	Read request pending No read request pending
b5	1 0	Write request pending No write request pending
b6	1 0	Data in buffer Buffer empty
b7	1 0	Buffer full Buffer not full

b7	b6	b5	b4	b3	b2	b1	b0

TABLE II—Interface unit status word

INTERFACE UNIT

Each processor node of Star architecture is connected to the dual baseline interconnection network by means of an interface unit. The interface unit has four ports, two active and two passive. Each of the dual baseline networks is connected to one active and one passive port. Only active ports can initiate a path connection request; once a path is connected, however, both the master and slave processor nodes can transmit or receive data.

The functions of the interface unit include providing the necessary handshaking between the processor node and the

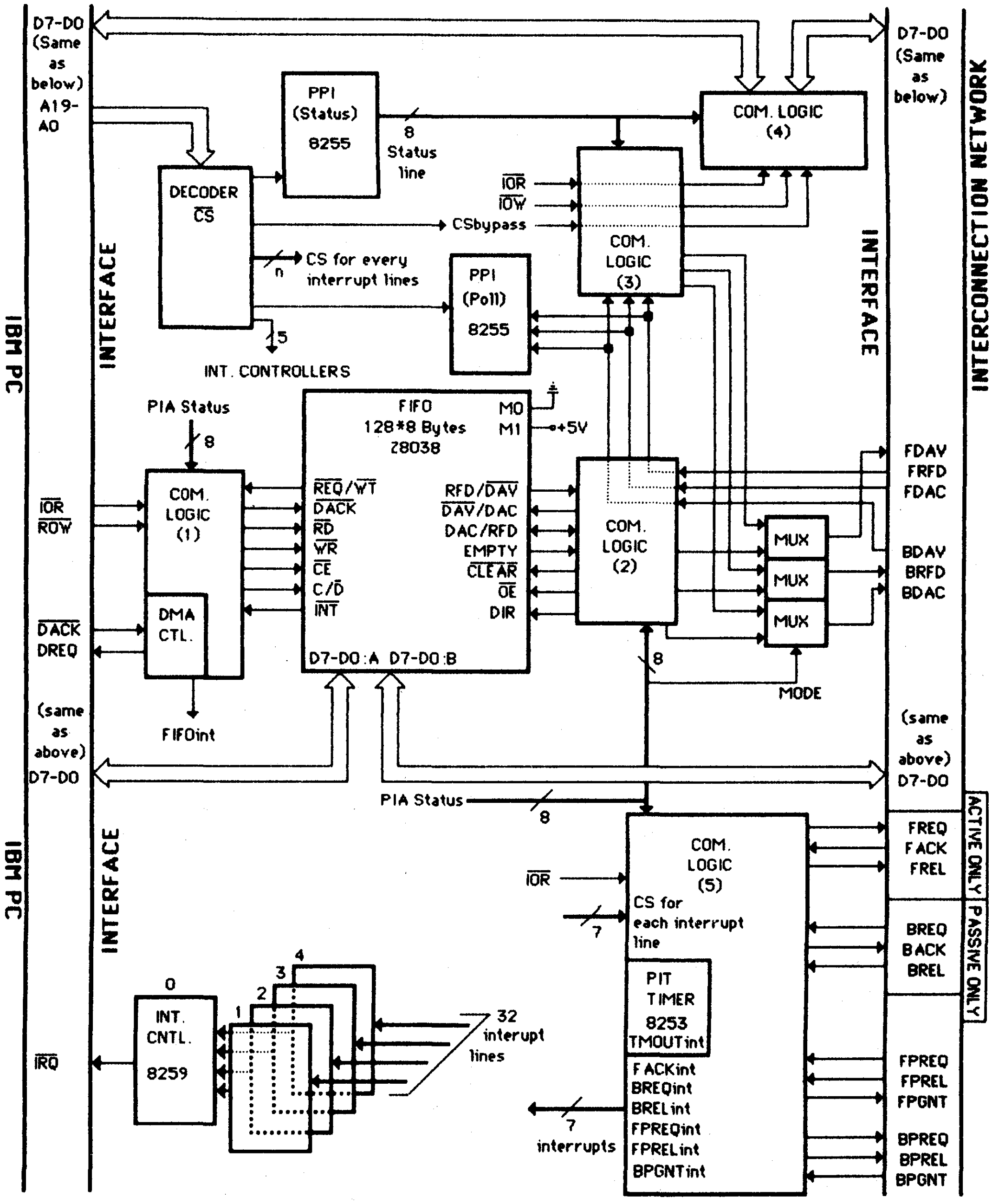


Figure 7—Block diagram of the interface unit

interconnection network, setting up and configuring a path between two or more processor nodes upon a request from a master processor node, and monitoring the data transfer once the path is set up.

The design of the interface unit is optimized in such a way as to take the overhead of path setup and configuration and data transfer away from the processor node so that the processor node can execute its own task with the least amount of interruption from the interface unit. The Direct Memory Access (DMA) and interrupt lines of the processor node are used for data communication and transfer. A control word and a status word are used for communication between the processor node and the interface unit. Through the different bits in these words, the interface unit and the processor node will "know" each other's state. Tables I and II show the function of each bit of the control and status words, respectively.

Figure 7 is a block diagram of the interface unit to be used in conjunction with an IBM PC/XT as a processor node. The main elements used in the interface unit are:

1. Intel 8237 DMA controller. This DMA controller is cascaded to the 8237 within the IBM PC/XT to give the

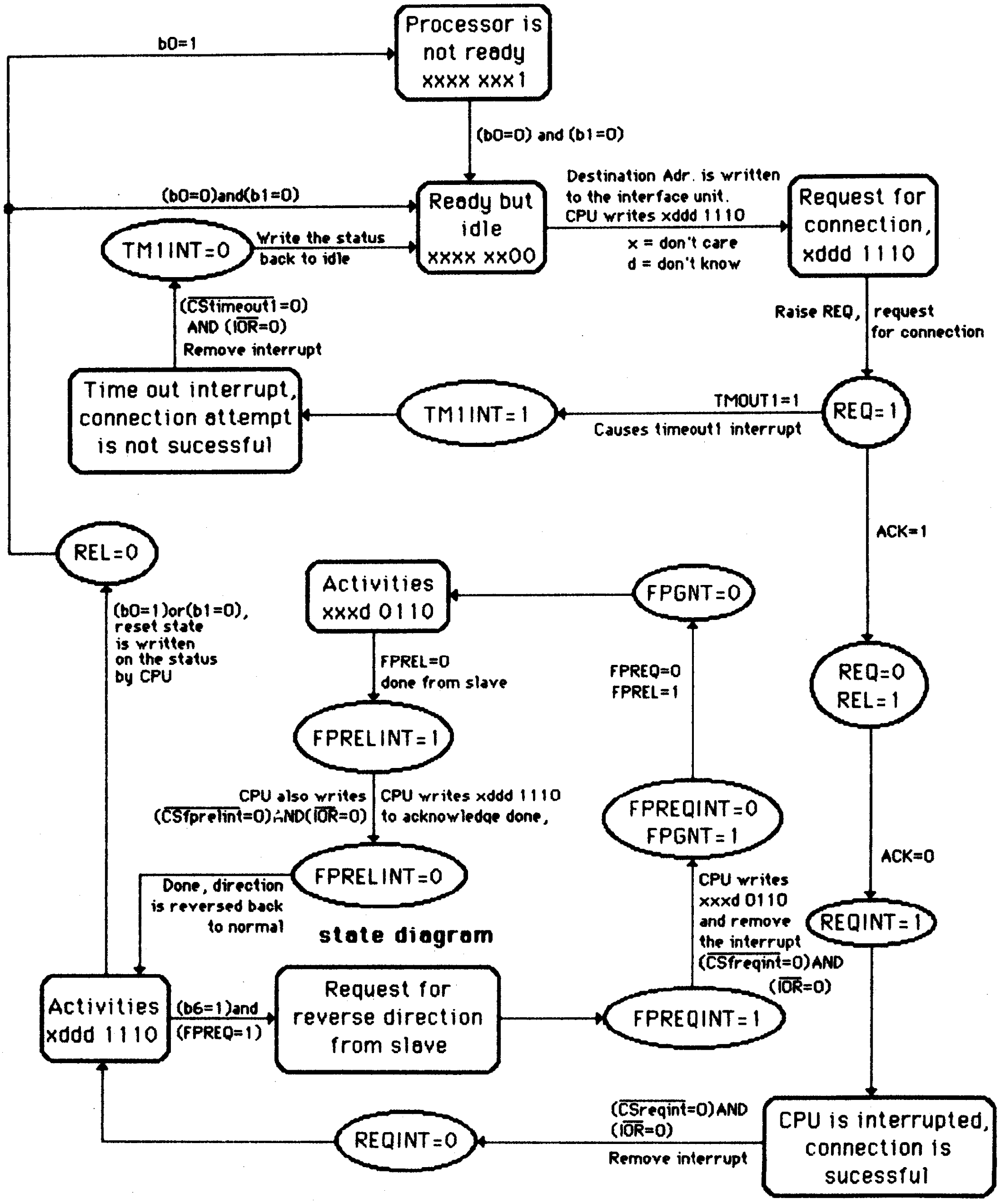

Processor is not ready xxxx xxx1
b0=1
(b0=0) and (b1=0)
(b0=0)and(b1=0)
Ready but idle xxxx xx00
Destination Adr. is written to the interface unit. CPU writes xddd 1110
x = don't care
d = don't know
Request for connection, xddd 1110
TM1INT=0
Write the status back to idle
(CStimeout1=0) AND (IOR=0) Remove interrupt
Raise REQ,
request for connection
Time out interrupt, connection attempt is not sucessful
TM1INT=1
TMOUT1=1
Causes timeout1 interrupt
REQ=1
ACK=1
REL=0
Activities xxxd 0110
FPGNT=0
(b0=1)or(b1=0), reset state is written on the status by CPU
FPREL=0 done from slave
FPREQ=0 FPREL=1
REQ=0 REL=1
FPRELINT=1
CPU also writes (CSfprelint=0)AND(IOR=0)
CPU writes xddd 1110 to acknowledge done,
FPREQINT=0 FPGNT=1
ACK=0
Done, direction is reversed back to normal
FPRELINT=0
CPU writes xxxd 0110 and remove the interrupt (CSfreqint=0)AND (IOR=0)
REQINT=1
state diagram
Activities xddd 1110
(b6=1)and (FPREQ=1)
Request for reverse direction from slave
FPREQINT=1
REQINT=0
(CSreqint=0)AND (IOR=0)
Remove interrupt
CPU is interrupted, connection is sucessful

Figure 8—Path set up, release, and configuration state

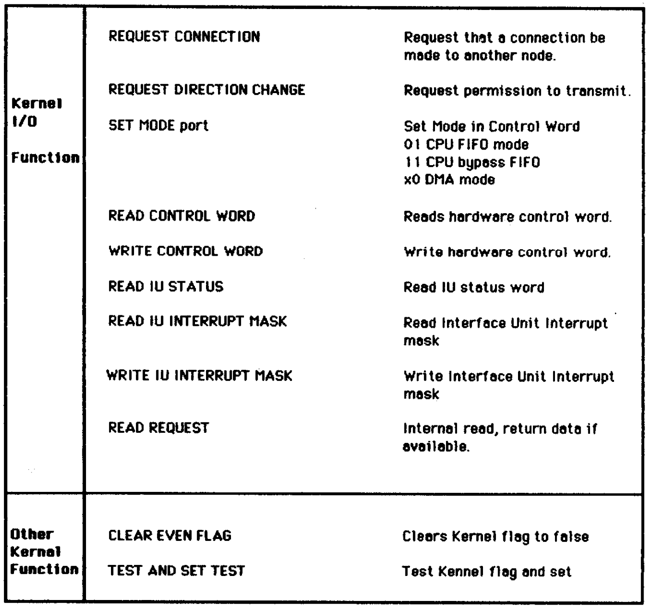
interface unit four DMA lines, one for each port. (Only 2 DMA channels are left unused on the I/O channel of the IBM PC/XT.)

2. Zilog Z8038 FIO (FIFO). This chip, used during data transfer to buffer the data, facilitates high speed data transfer and prevents bottlenecking in case one of the processor nodes is unable to process at the speed that the data is transmitted.
3. Intel 8259 Programmable Interrupt Controller. This chip is used to expand the interrupt lines of the IBM PC/XT. Interrupt lines are used for communication with the processor node.
4. Intel 8255 Programmable Peripheral Interface (PPI). This chip is used for generating the handshaking signals between the interface unit and the dual baseline interconnection network.
5. Intel 8253 Programmable Interval Timer (PIT). This chip is used for counting the timeout interrupt conditions.

Figure 8 shows a state diagram of the path setup and configuration stages. If a processor node wants to request a path connection to another processor node, it will send the destination address to the interface unit and write the appropriate control word (CW) on the PPI register. The processor node continues with other tasks while the interface unit takes the appropriate action to set up and configure the requested path. The destination address is sent to the data plane of the switching network. The interface unit will raise the REQ line and wait for an ACK signal from the slave processor node. If none is received and a timeout occurs, the interface unit issues a timeout interrupt. Upon receiving the timeout interrupt, the processor node writes the idle control word and requests path connection at a later time. If an ACK signal is received from the destination, the master interface unit raises the release line (REL). The interface unit also takes care of initializing path direction by generating the necessary handshaking signals. After the path is setup and configured, the interface unit issues an interrupt, notifying the processor node that the path is successfully configured. Once the master processor node finishes transmitting, it releases the path by clearing the interface unit control word.

Speed is an important factor in the Star network. One of the fastest methods of data transmission using the IBM PC/XT is DMA. Because the receiving processor node might not be able to process the transmitted data at the speed they are sent, an FIFO (Z8038) chip is used to buffer the incoming data. The Z8038 FIFO chip uses three-wire asynchronous handshaking signals: DAV (Data Available), RFD (Ready For Data), and DAC (Data Accepted). These signals are compatible with the IEEE 488 standard.

In some cases, DMA results in too much overhead. For such cases, a bypass mode is furnished after path setup and configuration; the FIFO and DMA are bypassed and the master processor node talks to the network directly through its data bus.



Kernel I/O Function	REQUEST CONNECTION	Request that a connection be made to another node.
	REQUEST DIRECTION CHANGE	Request permission to transmit.
	SET MODE port	Set Mode in Control Word 01 CPU FIFO mode 11 CPU bypass FIFO x0 DMA mode
	READ CONTROL WORD	Reads hardware control word.
	WRITE CONTROL WORD	Write hardware control word.
	READ IU STATUS	Read IU status word
	READ IU INTERRUPT MASK	Read Interface Unit Interrupt mask
	WRITE IU INTERRUPT MASK	Write Interface Unit Interrupt mask
	READ REQUEST	Internal read, return data if available.
Other Kernel Function	CLEAR EVEN FLAG	Clears Kernel flag to false
	TEST AND SET TEST	Test Kennel flag and set

TABLE III—Kernel functions

OPERATING SYSTEM KERNEL AND INSTRUCTION SET

Operating System Kernel

The OS kernel of Star architecture is currently being implemented as an extension to Personal Computer Disk Operating System (PCDOS), and is loaded at system-boot time. The kernel provides the necessary extensions to PCDOS for system-wide communication and resource management. Star kernel functions are used in the same manner as PCDOS system-calls, and are called through software interrupts. This portion of the paper will consider only the kernel functions which are necessary for inter-port communication.

The I/O port software is designed to be compatible with PCDOS. The kernel provides an I/O driver for each of the four ports at the node. These drivers are written and installed according to the guidelines provided in the IBM DOS Technical Manual. The advantages of using this format for the port drivers are that the drivers are automatically installed at boot time, and PCDOS file manipulation-calls may be used to directly address the ports. The second of these advantages assumes that the user configures the port before using it.

The interface unit has a number of modes of operation for the anticipated uses of Star. These include interrupt-driven data transfer, selective data acknowledgment for the SIMD environment, DMA, and non-DMA transfers. The Star I/O-calls described in Table III include three instructions that are unusual in that they initiate a request, but do not wait for the result. These instructions, *Request Direction Change, Request Connection,* and *Read Request,* are implemented in this manner because they are I/O-dependent and might result in long waiting times. It is up to the user process to determine whether the requested task has been completed. This could be done in a straight polling of the port status, but the kernel provides a utility to simplify this task. In its data area, the OS kernel maintains a set of event flags, and provides two instructions for their manipulation, *Test and Set Flag,* and *Clear Flag.* Each of the I/O ports is associated with an event flag that is cleared whenever the Port Device Driver processes a hardware interrupt. The use of these event flags assumes that one process controls access to the Port Device Driver.

The flowchart in Figure 9 illustrates how a process and a Port Device Driver cooperate in configuring an I/O port. In this example, the process needing configuration polls the event flag to determine whether an interrupt has occurred. Ultimately, a Starnet utility will be provided to allow a process to wait for an event while other processes are allowed to run.

Once port configuration has been completed, a process may use the PCDOS I/O calls to read from and write to the port. These calls are particularly useful in transferring files from one node to another. This set of I/O functions is not intended to represent the final, complete set; still to be considered are synchronous read/write commands (these allow Starnet to operate in an SIMD mode).

Instruction Set

This subsection assesses the processor instruction set for Star. The current implementation of Star is going to use IBM

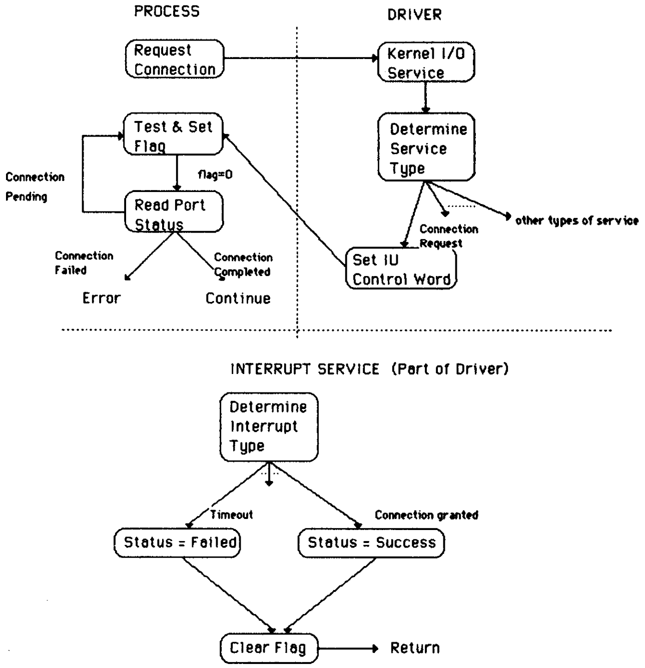


Figure 9—Example of port configuration

PC/XTs as the processor nodes of the system. These processor nodes will be programmed using the available tools of the XT. Because the XT is based on Intel's 8088 processor, the use of instruction set optimization techniques is not envisioned for the initial part of the project. The operation of the Star processor nodes will be simulated on the IBM PC/XT. Ultimately processor architecture will be tuned so that the Star type instructions are executed optimally.

In general, implementation of a virtual machine instruction set on a host computer using the machine language instruction set of the host is called *simulation*. In the first part of our research, the virtual instruction sets of Star processor nodes are designed using Intel 8088 assembly code. This simulation has two benefits as far as processor organization is concerned: (1) it allows the designer to test a set of instructions that seem to be good candidates for the final system implementation. Based on the simulation results, the instruction will be augmented to handle higher level functions, or reduced to eliminate bottlenecks in the micro-operation; and (2) simulation allows the system programming stage to proceed smoothly without requiring severe changes during code migration to the final processor design.

Table IV shows a list of virtual instructions that have been identified for some of the applications of Star architecture. These instructions are invoked by calling a subroutine with a name similar to the presented mnemonics. These calls execute a set of assembly language routines which perform the required actions for the virtual machine.

RESOURCE ALLOCATION

The proposed resource management mechanism[5] is an object-based approach to structuring the resources of Star and raising

Mnemonic	Comments
BRCA	Broadcast instruction/data. This instruction is used during the SIMD mode of operation.
RSNT	Reset the network. This command is used to initialize the system either for synchronization or for power up stage.
STPR	Set priority for the path. This handles conflicts at the switching elements.
PARL	Path release. This command informs the master processor node that the slave processor node is done with the path.
PARG	Path request. The processor node wants to change the direction of the path.
PAGR	Path grant. This instruction grants the processor node the path for transmission.

TABLE IV—Virtual instruction set for the Star

the level of abstraction of the resources. The concept of an object-based approach is to decompose the entire resource management problem into a set of various object types of the resources, and a set of operations that manipulate the objects.

In the Star system, a processor node is chosen as the manager responsible for the resources of the system. Processor nodes not allocated are organized into clusters of different sizes according to certain rules. In response to a request for resources, a cluster from the cluster pool is chosen and transformed into a subsystem configured in the form of various topologies such as linear array, loop, binary tree, etc., to match the computation graph of the task. The subsystem can also be autonomous, whereby a node is chosen as the manager to govern the resources within the encapsulation and schedule tasks generated within the subsystem or received from outside without the intervention of higher level managers. This communication among objects and the manager is dynamically configured using idle processor nodes and communication links on the control tree, without using extra system hardware.

The set of operations performed on the objects is composed of the following procedures:

1. Resource Restructure Procedure (RRP)—to organize free processor nodes into clusters of various sizes.
2. Messenger Assignment Procedure (MAP)—to set up the control tree connecting all the objects currently present in the system.
3. Resource Allocation Procedure (RAP)—to allocate a cluster to form a subsystem.

Figure 10 illustrates how these procedures are cooperatively used to manage resources.

1. In its present state (Figure 10(a)), the Star of size 32, contains three subsystems:
   (a) mesh, {16, 17, 18, 19, 20, 21, 22, 23},
   (b) linear array, {2, 3, 4, 5, 6, 7}, and
   (c) loop, {30, 31},

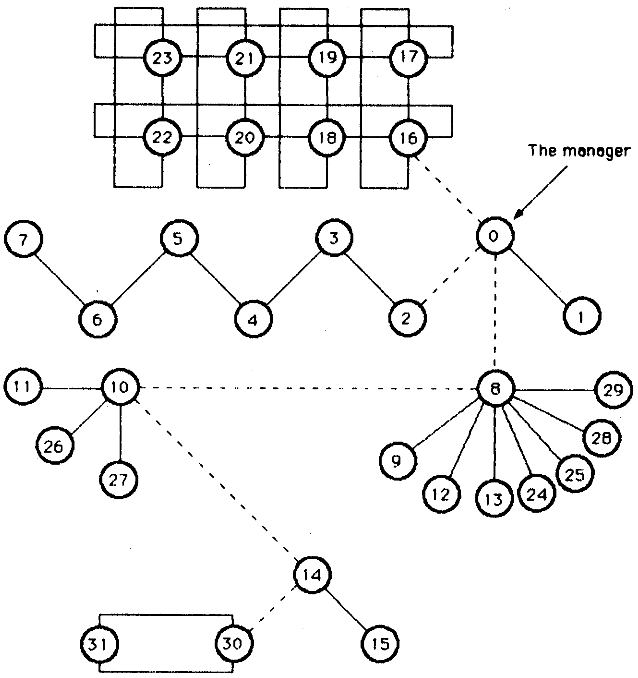


Figure 10—(a) Resource allocation example

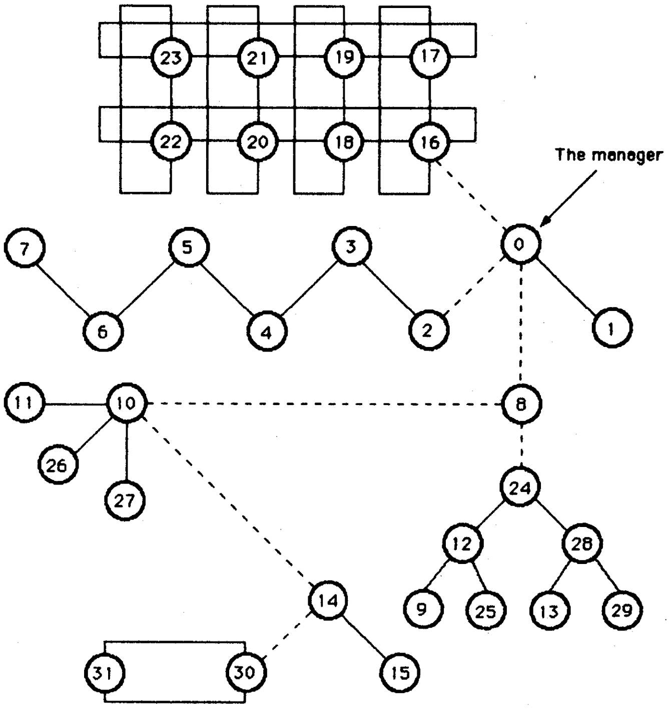


Figure 10—(b) State after a binary tree is configured

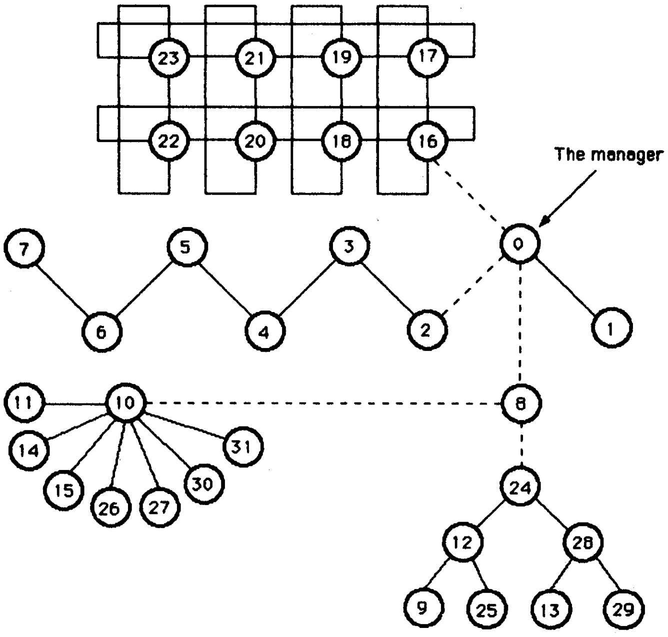


Figure 10—(c) Final state after loop (30,31) is released

and four clusters:

(a) {0,1},

(b) {8,9,12,13,24,25,28,29},

(c) {10,11,26,27}, and

(d) {14,15}.

2. A request for a binary tree of size 7 arrives.
3. The RAP is invoked and finds that cluster {8,9,12,13, 24,25,28,29} can accommodate the binary tree. It then forms the subsystem with a binary tree of size 7, as shown in Figure 10(b).
4. Now suppose loop {30,31} is released.
5. The manager invokes the RRP to restructure the clusters and the control tree. The final state of the system is shown in Figure 10(c).

SUMMARY

A prototype machine using eight IBM PC/XTs as processor nodes is being built to simulate an 8-node Star system. Currently, the design and implementation of interconnection networks, interface unit, operating system, and instruction set is proceeding well. Problems related to compiler design such as task partitioning are also being attacked. The virtual machine will be used as a testbed for developing software and application programs. Some applications have already been designed and theoretically evaluated.[6,7]

REFERENCES

1. Wu, C., T. Feng, and M.-C. Lin. "Star: A Local Network System for Real-time Management of Imagery Data." *IEEE Transactions on Computers*. C–31 (1982), pp. 923–933.
2. Wu, C., and T. Feng. "On a Class of Multistage Interconnection Networks." *IEEE Transactions on Computers*. C–29 (1980), pp. 694–702.
3. Lee, M., and C. Wu. "Performance Analysis of Circuit Switching Baseline Interconnection Networks." *Proceedings of the 11th Annual Symposium on Computer Architecture,* New York: Computer Society, 1984, pp. 82–90.
4. Lee, M., E. Fiene, C. Wu, G. Brown, and N. Bagherzadeh. "Network Facility for a Reconfigurable Computer Architecture." *Proceedings of 1985 Distributed Computing Symposium*.
5. Lin, W., and C. Wu. "An Object-based Resource Management Mechanism for a High Performance Distributed Computing System—Star." *Proceedings of COMPSAC '84*. Chicago, 1984, pp. 208–216.
6. Ting, K-C., and C. Wu. "Versatile VLSI Fast-Fourier Transform Processor." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 53), 1984, pp. 151–162.
7. Chen, A. C., and C. Wu. "Optimum Solution to Dense Linear Systems of Equations." *Proceedings of the 1984 International Conference on Parallel Processing*. Shanty Creek, 1984, pp. 417–424.

System and software security via authentication handshake in EPROM

by DAVID HOFF, ALAN FOLMSBEE, and LAWRENCE LETHAM
INTEL Corporation
Folsom, California

ABSTRACT

A 128-Kbit EPROM is described that can protect systems and software from unauthorized use. The KEPROM™ keyed-access EPROM will not allow read access to its array until an authentication handshake has been completed. The handshake uses two EPROMs and involves encryption of pseudorandom numbers using a secret 64-bit key stored in each part. Security is provided by requiring the two EPROMs to contain the same 64-bit key before either will grant read access. Because the secret key is known only to the system manufacturer, the use of unauthorized software with authorized systems, and vice versa, is prevented.

KEPROM™ is a trademark of INTEL Corporation.

The cost of software piracy is growing at an enormous rate. Disk-based software is routinely copied without a thought given to legality. Computer clubs often purchase one copy of a particular software package and distribute unlicensed duplicates to members. These and many other forms of prohibited software distribution account for the growing dilemma over piracy now faced by the software industry.[1] With the ever-growing proliferation of computer equipment, software protection is needed to stop the spread of piracy.

Attempts to provide copy protection of disk-based software have proved feeble. For every security mechanism developed, means to break it follow shortly thereafter. Programs such as LOCKSMITH, from Omega Software Products, Inc., have foiled copy-proof disks. Conservative estimates of sales losses due to software theft can easily run as high as 30% of legitimate sales.[1] This major loss to a billion-dollar-a-year industry can no longer be tolerated. Effective, inexpensive solutions are needed for this problem; otherwise, much of the incentive to produce a large selection of high-quality software will be gone, and the computer industry as a whole will suffer.[1]

A new IC nonvolatile memory device has been developed to provide software and system security. The KEPROM keyed-access EPROM erects a barrier between software and pirate, protecting systems and software alike from unauthorized use. It is completely pin-compatible with the standard 128-Kbit EPROM and contains 128 Kbits of EPROM memory, a secret 64-bit key, a pseudorandom number generator (PNG), and encryption circuits. When the EPROM is first powered up, its array is locked and cannot be read. In order to grant read access, an authentication handshake must be performed between two EPROMs, each containing the same 64-bit key. If the two keys do not match, the handshake will fail and neither chip will grant read access. Only if both chips contain the same secret key will read access be granted. The correct key is known exclusively by the system manufacturer and his designated parties. Security is maintained by requiring both authorized chips to be present in the system before either will unlock.

Figure 1 shows a system using two EPROMs. The system EPROM, known as the *recipient*, contains vital operating system subroutines, without which the computer will not function. This EPROM is attached permanently to prevent attempted substitution by pirates. The chip may be epoxyed to the circuit board or a metal shield may be riveted over it. Another technique is to place a seal on the system EPROM that must be broken to remove the chip. The customer is instructed that breaking the seal results in a voided warranty. This method of security is inexpensive, and allows for easy upgrading by the dealer. Few customers are willing to subject a system to the physical damage that results from removing a riveted or epoxyed chip, nor are they willing to void a warranty by breaking a seal. With the system EPROM permanently attached, only authorized EPROMs containing the correct 64-bit key can be run on the system. This allows the manufacturer to protect the system and the software from prohibited use.

The removable cartridge contains one or more authorized software packages stored in its EPROM. The *originator* EPROM contains any one of the many popular software packages available today, including word processing, accounts payable and receivable, or spread-sheet. To run a particular package the user need only plug the correct cartridge into the system. The cartridge software calls certain vital subroutines in the system EPROM, requiring the application to run only on systems containing the EPROM with the correct key.

When power is first applied, both EPROMs are locked; their arrays cannot be read, except for locations 0 through 01FF and 3FF0 through 3FFF hexadecimal. These 528 locations are provided for bootstrap code and can always be read. The microprocessor begins execution out of the system EPROM bootstrap code area. No program code is executed out of the cartridge until it has been shown to be authorized to run on the system, via the handshake. The handshake is two-way so that both sides are authenticated to each other. It begins with the originator EPROM, located in the cartridge, generating a 32-bit pseudorandom number. This number is transferred serially by the microprocessor to the recipient.

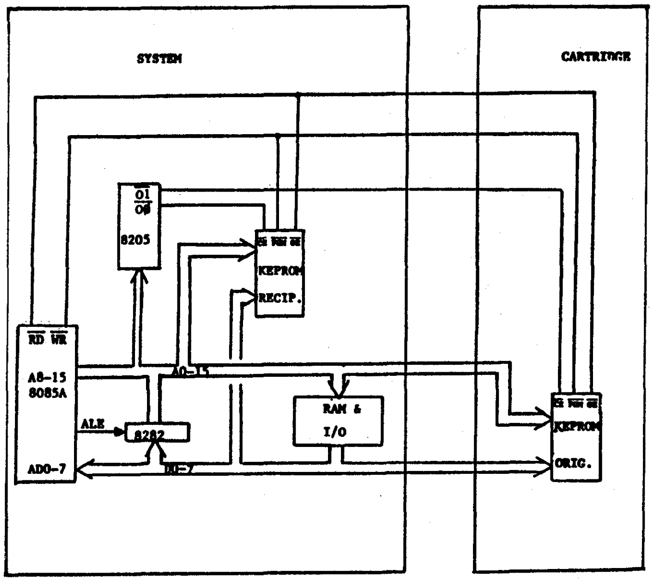


Figure 1—KEPROM keyed-access EPROM system

Eight dummy read cycles are then performed on the recipient to load the secret key from the array into the encryption circuits. Each chip scrambles the number using its own secret key. The encryption algorithm is a complex, nonlinear substitution–permutation algorithm modeled after the Data Encryption Standard (DES).[2] After the encryption is complete, the 32-bit result from the recipient is transferred back to the originator, where it is internally compared against the originator's own encryption result. If the two EPROMs contain the same secret key, then the encrypted outputs will match and the originator will grant read access to its entire array. At this point, the system has been authenticated to the cartridge; it is authorized to run the software.

To provide protection against running unauthorized software on the system, a second half of the handshake is performed to authenticate the cartridge. The two EPROMs reverse roles, with the system EPROM now generating the pseudorandom number. Upon successful completion of the second half, the cartridge has demonstrated that it is authorized to run on the system. The system EPROM will grant read access and execution of the software package may proceed at full speed. The total time for the two-way handshake ranges from 100 milliseconds to 10 seconds, depending on the setting of a programmable delay counter. The architecture of the EPROM provides a high level of security at no penalty in system performance. Once unlocked, the KEPROM device operates as a standard EPROM.

The two-way handshake is a powerful feature of the KEPROM device in that it protects both the software and the system from unauthorized use. The random-number encryption scheme allows two initially unfriendly devices to authenticate each other as containing the same secret key, without either side revealing what the key is. If the key were exchanged, a pirate could decode it after one unsuccessful handshake—and security could be violated. The PNG makes every handshake different, which prevents memorization of the correct sequence for authorization. Each device can authenticate the other while maintaining its own security. This makes the EPROM useful in numerous friend–foe applications, including those not directly related to the security of software and systems.

In the example shown in Figure 1, the system EPROM is able to authenticate any authorized software cartridge EPROM containing the correct 64-bit key. There are many applications in which it is desirable to have the system EPROM authenticate EPROMs containing different keys. Versatile personal computers must be able to run numerous software packages from many different distributors, all of whom want to keep their software secure from theft. Sharing one key among all software vendors would significantly degrade security for each program. To allow flexibility when securing several software applications using different keys, the key manager mode has been added. When the EPROM is programmed as a key manager half of its array becomes storage for 1024 keys. The other half is used to store data and programs. Once locked as a key manager, the keys can never be read at the outputs again. The key manager is typically used in conjunction with other nonkey managers. The one key manager is able to authenticate up to 1024 different EPROMs, each with its own key.

In the example, the system EPROM could be a key manager able to authenticate different EPROMs. Each authorized application would have its own key, allowing it to authenticate to the system key manager. Different software vendors would be allocated private sets of keys, which would protect their security. Thus, the system manufacturer also has the ability to control the software run on his system. The manufacturer could even have several versions of key manager EPROMs for the same system hardware. The level of sophistication for a given model could be based on what keys were programmed into its key manager. A complete system EPROM would be programmed with the full set of keys; the key manager in the economy system would have fewer. A low-cost system designed to run only word processing and spread sheet software would have its key manager programmed with keys for those specific packages only. Therefore, a pirate or an unscrupulous customer would be unable to remove the key manager, thus preventing the use of pirated software on the system.

When a customer wants to upgrade his system legally, the system board is traded in for one with a key manager containing more keys. If a seal is placed on the key manager, the dealer simply replaces the key manager alone. A new seal is applied to ensure that no tampering occurs with the new key manager.

Another benefit of the key manager mode is that several different software cartridges can be authenticated to the system simultaneously. Each cartridge can contain valuable programs or data. The system shown in Figure 2 has the key manager authenticated to four cartridge EPROMs. The handshake is performed four times; once for each cartridge.

To ensure that the correct key is loaded out of the key manager array during the handshake, a special register has been included on the EPROM. The nonkey manager has this register programmed with the address of its key in the key

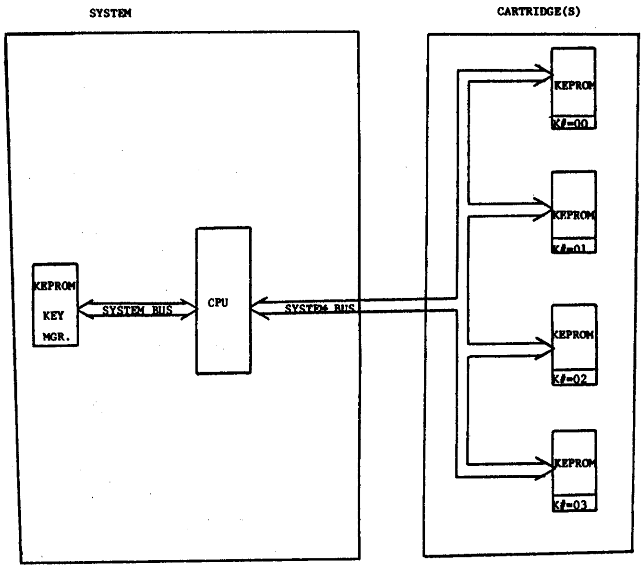


Figure 2—Key manager architecture

manager array. The key address is read by the microprocessor and is used to load the correct key from the key manager for encryption. If the wrong key is loaded or the EPROM is not authorized to be used on the system, then the keys will be different, resulting in a failed handshake.

Computer users who are involved in software pirating have varied levels of sophistication. The casual copier duplicates a disk containing a popular word processor or video game. Mostly, these copiers have minimal technical skill and are able to copy software simply because it is easy to do so. Many business owners and hobbyists fall into this category. Uncopyable disks are easily duplicated by these casual copiers once the special trick is learned.[1]

The EPROM will severely inhibit the casual copier who attempts to use an EPROM programmer to copy the KEPROM memory. This attempt will not work because the EPROM is locked and its contents are unreadable. Any other such attempts will likewise fail. Even if the EPROM software is copied from an authenticated system by monitoring the busses, it cannot be run on another authenticated system. If it is stored on disk, the disk cannot perform the handshake needed to unlock the system EPROM. Storing the code in another KEPROM memory is likewise futile without the secret key. The casual copier soon will give up and purchase a legitimate copy of the software.

The next level of software pirate may be termed the technician copier. These pirates have a basic understanding of electronics and computer architecture. They also have power supplies, development systems, and test equipment to help them. Most software pirates fall into one of these two categories. The technician copier can put the cartridge in an authorized system and allow the handshake to occur, unlocking both parts. He then monitors the address and data busses and, in time, discovers all of the data stored in the EPROMs. But he still does not know the secret key and—like the casual copier—cannot use the software on an authorized system. Without the correct key, his bogus cartridge will fail the handshake and the system will stop execution. The technician copier is thereby prevented from peddling his software to authorized system users.

The most sophisticated copiers have access to IC reverse-engineering lab equipment. In the lab, the EPROM can be dissected in an effort to determine the secret key. The success of this attack is highly unlikely, because the key is stored only as a charge in EPROM cells. It cannot be seen by examining the die. Even if the attack is successful, the cost would run into the millions for lab equipment alone. This is far beyond what most copiers can afford and it prevents any effective attack on the EPROM. In addition, any organization large enough to effectively reverse engineer the EPROM also can be sued for copyright infringement. The EPROM is thus able to protect valuable software from pirates. Above all, the EPROM removes the economic benefit of theft that has allowed software piracy to become so widespread.

The encryption and PNG circuits critically affect EPROM's security. The pirate foiled in his attempt to break into the EPROM will most certainly attack these two important circuits. The PNG must generate truly random-looking numbers. If the output were to repeat too frequently or degenerate into a sequence, the EPROM could fall prey to cryptographic attack. The encryption algorithm also is subject to attack by pirates, who can purchase an EPROM with the desired software and feed it chosen random numbers. The encrypted outputs could be analyzed to determine the key. This form of attack is known as a chosen-text attack and is the most difficult to withstand.[2] Such considerations were taken into account in the design of these two vital circuits.

When considering the analysis of a sequence of numbers to determine their degree of randomness, one is stuck by the question of how one can determine whether a sequence is random at all. Any sequence can be random; a sequence of zeroes has a finite probability of being output from a random-number generator. The mathematician, Donald Knuth, in his book *Seminumerical Algorithms,* explores this question at great length.[3] Among his major points is that a sequence can be evaluated as random using properties that are present in ideal random sequences.

The following seven tests were used to evaluate 900,000 random numbers collected from ten EPROMs. First, a search was performed to determine the most common pseudorandom number generated. The number 00000000 appeared 18 times. 128 numbers appeared between 2 and 11 times, resulting in more than 99.97% of the numbers generated appearing only once. The low percentage of repeat numbers generated hinders attacks by a pirate who monitors the PNG output for multiple occurrences of the same pseudorandom number. A listing of the numbers occurring three or more times is given in Table I.

In the second test, the correlation coefficient between consecutively generated pseudorandom numbers was tested to see if the next output generated is a function of the present one. The value of 0.00238 was impressive, indicating no such connection.

Next, the correlation coefficient between the two words within each pseudorandom number was calculated to see if the components of the number are related. The value of 0.0164 indicates no relationship.

In the fourth test, the correlation coefficient of two pseudorandom number lists beginning with the same value was calculated. This experiment examined the output for sequences. The value of −0.00936 strongly indicates that a sequence is not being generated.

In the fifth experiment, the correlation coefficient between

TABLE I—Pseudorandom numbers occurring three times or more

Number	Occurrence Count
00000000	18
20000000	9
80000000	8
40000000	5
10000000	4
28000000	3
9C800000	3

TABLE II—Nibble occurrence counts

Nibble value	Occurrence count
0	324786
1	319417
2	311149 MINIMUM
3	322481
4	312293
5	317407
6	312401
7	326608 MAXIMUM
8	318696
9	315960
10	319898
11	318512
12	319930
13	320164
14	326456
15	325890

lists from two different EPROMs was tested to see if the outputs were mathematically related. The result of −0.0011 indicates no such relationship.

Next, the percent of 1 bits generated was shown to be 50.15%, which is close to the ideal value of 50%.

Finally, a frequency-of-occurrence table for nibbles was developed. This is shown in Table II. The maximum disparity in counts for nibbles is only 5%, indicating an even distribution of values generated.

The results obtained in these tests were quite impressive and more than adequate for the intended operation. The count of 18 for the most common number results in an average spacing of 50,000 between repeats for that number. The design goal was 20,000 numbers between each repeat occurrence of any number. The "birthday problem" theory predicts about a 50% chance of repeating a 32-bit number after 65,546 samples.[2] The PNG described here has a more than 50% chance of repeat for that sample size, yet the results are comparable. Although not as good as a perfect random-number generator, the results obtained are sufficient to deter pirates from monitoring the PNG output for repeating random numbers.

The interval between repeating pseudorandom numbers is perhaps the most critical test of the PNG. If a particular output of the PNG occurs too frequently, a pirate may be able to authenticate a counterfeit cartridge to the system. He could simply capture a random number and its encrypted output. He would then repeatedly reset the system EPROM until his random number recurred. The known encryption result would be used as the response, enabling the system EPROM to be unlocked. The pirate's unauthorized cartridge would now be authenticated to the system, resulting in a breach of security. The high value for the interval between repeats makes this line of attack impractical. The process of waiting for a specific output of the PNG would have to be repeated every time the system was powered up, a performance penalty too large for most custômers. The PNG has a high degree of randomness; more than sufficient for the security application intended. A detailed explanation of the PNG's internal operation is described in Reference 4.

Copies of proprietary software alone are not functional in an authenticated system; the pirate must obtain the secret key stored in the part. He may either try to analyze the encryption algorithm or perform some form of IC reverse engineering to determine the key. The latter technique is beyond the financial means and technical abilities of the vast majority of pirates and copiers. Because the key is never revealed at the outputs, the pirate is forced to attempt analysis to determine it. The pirate can obtain a recipient EPROM containing the desired software, feed it random numbers and collect the encrypted outputs. The numbers can then be analyzed in an effort to determine the key. To ensure that this analysis will fail, extreme care was used in the design of the EPROM encryption algorithm.

Principles from the DES, along with additional security concepts, were included in the design of the algorithm.[5] Figure 3 shows a block diagram of the EPROM encryption algorithm. The random number is loaded into the accumulator, and the key register holds the lower 32 bits of the key. The encryption of the accumulator value proceeds using the lower 32 bits of the key to control the operation. When the algorithm is complete, the upper 32 bits of the key are loaded into the key register and the process is repeated. The random number is encrypted twice using both halves of the key before being sent to the outputs.

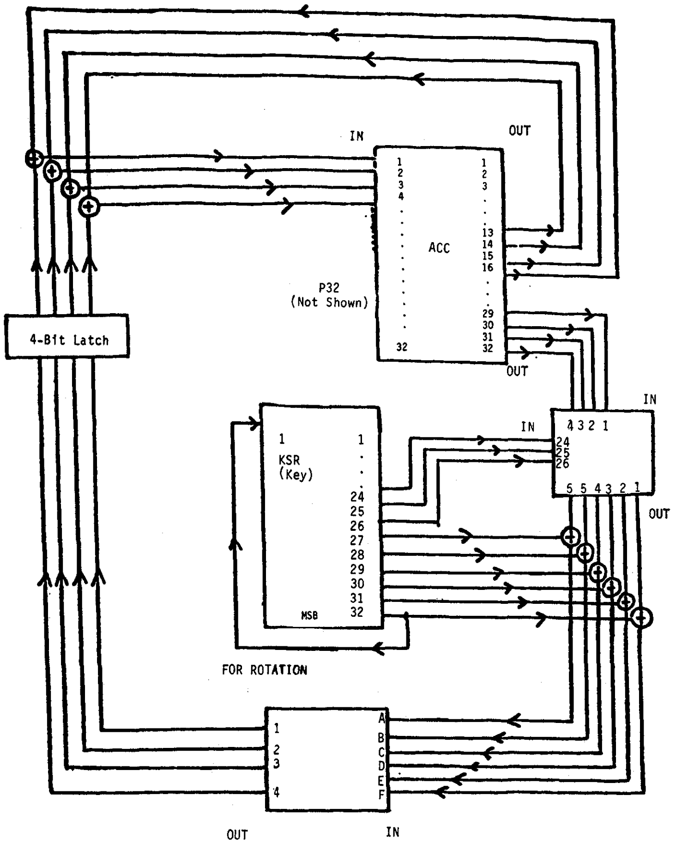


Figure 3—Encryption algorithm block diagram

The P-box performs a nonlinear permutation of accumulator bits 29 through 32 using key register bits 24 through 26. The S-box is a substitution ROM with 6 address inputs and 4 outputs. The algorithm performed using both 32-bit halves of the key proceeds as follows:

1. Latch in the 4-bit result propagating from the accumulator and key register.
2. Shift the accumulator 4 bits to the left; disregarding bits 29 through 32.
3. Load the 4 bits from the latch into the accumulator.
4. Repeat steps 1 through 3 eight times.
5. Perform a 32-bit permutation of the accumulator. The permutation circuit (not shown) performs a wire crossing of all the bits in the accumulator. No two bits in the same nibble before permutation remain grouped together after it is complete. This foils the pirate analyzing the algorithm in terms of nibble operations.
6. Rotate the key register 7 bits to the left. The value in bit 32 is moved to bit position 7.
7. Repeat steps 1 through 6 32 times.

The substitution ROM is similar to the S-boxes in DES. The 32-bit accumulator permutation is likewise modeled after a function in the standard. Although there are similarities, the algorithm differs notably from DES for die size and security considerations. By using the current algorithm instead of a high-speed DES circuit 35,000 mm^2, or 70% of the total area, was saved. Additional modifications were made to ease concern about the security of DES. Thirty-two rounds, each performing a 32-bit permutation of the accumulator, are executed for each half of the key. DES has been criticized for performing only 16 encryption rounds.[2]

The EPROM algorithm is one-way, limiting the flexibility of the pirate in his attack. DES is a two-way algorithm, allowing both encryption and decryption of data. No parity bits are used in the key, so a true 64-bit key is available, as opposed to 56 in DES. Weaknesses have been found owing to partial linearity in the S-boxes,[2] the only nonlinear element in DES. The EPROM S-ROM was designed to be nonlinear. A change of a single bit in the address causes at least two output bits to change.

To heighten security, a special 6-bit permutation box was added. The 3 key bits into this P-box have a nonlinear effect on the output. By introducing the key using a nonlinear mechanism, security is enhanced. Because of this nonlinearity, the encryption is not invariant under complementation of plaintext, key, and ciphertext. The DES algorithm has been criticized for having this invariance, because it reduces the CPU time needed to do a brute-force search for the key. Using the key in a nonlinear way relieves security considerations about the DES key-scheduling algorithm.

Error propagation, the number of bits changing in the output for a single bit change in the input, is also an important property for encryption algorithms.[2] A single bit change in either the random number input or the secret key should cause many bits to change in the output. The P-box and S-ROM enhance error propagation; a single input bit change usually results in about half of the bits changing at the outputs. With the enhancements present in the KEPROM encryption algorithm, many of the apprehensions about DES have been resolved. This has been done with a significant saving of die area as compared with a literal implementation of the standard. The level of security provided by the implementation is more than sufficient to thwart both the casual copier and the sophisticated pirate.

CONCLUSION

A new single-chip solution to software piracy has been described. This EPROM allows systems and software to be secured from unauthorized use, enabling the rightful owners and inventors of new technology to receive their just rewards. The KEPROM component contains 128 Kbits of EPROM memory for program storage, encryption circuits, and a pseudorandom number generator allowing it to determine if read access is to be granted. Only access by authorized systems is allowed, protecting the valuable software contained in the chip. The product is completely pin-compatible with the standard 128 Kbit EPROM and will be easily incorporated into future designs.

ACKNOWLEDGMENTS

The authors are grateful to Steve Domenik for formulating the KEPROM concepts, to design engineers Tai Nguyen and Emilio Yero, and to the layout designers, of whom the most notable are Steve Kiesler and John Zaferes. We also thank Robert Davis, Kurt Robinson, Scott Brock, Bill Spaw, Don Knowlton, Larry Palley, Ray Delissio, Dave Townsley, David Shrigley, and George Schneer for their support and work.

REFERENCES

1. Goldschmitt, M. "Confronting Software Theft." INFO 84 Conference. August 1984.
2. Hellman, M. E. *Cryptography and Data Security*. Palo Alto, Calif.: Hellman Associates, 1981.
3. Knuth, D. E. *The Art of Computer Programming* (2nd ed.). Reading, Mass.: Addison-Wesley, 1981, Vol. 2.
4. Folmsbee, A., D. Hoff, and L. Letham. "128K EPROM with Encrypted Read Access." *IEEE ISSCC Digest of Technical Papers*, 1985.
5. FIPS PUB 46. "Data Encryption Standard." January 1977.

Database machines: A survey

by GHASSAN Z. QADAH*
Northwestern University
Evanston, Illinois

ABSTRACT

This paper surveys a class of computer architectures that use parallelism to enhance the performance of relational database systems. These architectures are referred to as Database Machines (DBMs). This survey groups the database machines into a small number of classes according to a newly developed classification scheme. The classes are then reviewed, examined, and compared in light of current and future trends in processor and memory technologies.

* Ghassan Z. Qadah is currently visiting Institut Für Informatik—the Swiss Institute of Technology—Zurich, Switzerland.

1. INTRODUCTION

Many relational database systems have been implemented since E. F. Codd introduced the relational model of data in the early seventies.[1,2] Most of these systems have been organized as a complex software system, the database management system (DBMS), running on a large general-purpose von Neumann computer and provided with a friendly very high level language (a query language) for user interface.[3] This conventional design approach resulted in relational systems with low reliability, poor performance and inability to meet the demands for large processing power and throughput to fulfill the current and anticipated increases in database sizes and usage.

The limitations of the conventional approach, the fact that many of the operations implemented by the relational database system lend themselves well to parallel processing, and the continuous reduction in the memory and processor costs inspired a new approach to the implementation of relational systems. This replaces the general-purpose von Neumann computer with a dedicated Database Machine (DBM) which is tailored for non-numeric data processing and, in most cases, utilizes parallel processing to support some or all the functions of the DBMS. The new approach claims to improve the system's reliability through software complexity reduction, and enhances the system's performance through specialization, increased parallelism, and increased processing power.

During the past 10 years many DBMs have been proposed, some of which have also been prototyped[4,5,6,7,8] and few others have been commercialized.[9,10] Most of DBMs proposed so far, are organized as a backend machine to one or more general purpose computer(s) known as host(s). The host-backend organizational concept is attributed to Canaday.[11] In such organizations, the host is responsible for interfacing the user to the DBM, and the DBM itself is responsible for the database access and control.

This paper surveys the DBMs proposed to date. The emphasis of this survey is placed on those machines designed to support comprehensive relational systems. Less emphasis is placed on those machines designed only to speed up the execution of a specific operation of the relational system. Section 2 overviews the structure of a relational database and the most important operations defined on such a structure. It also examines briefly the nature of the queries through which a user gets access to the relational database. The third section presents a novel classification scheme for the DBMs. Guided by this scheme, Section 4 overviews the DBMs proposed to date. Finally, Section 5 presents a critical look at the surveyed machines.

2. RELATIONAL DATABASES

A relational database[1] consists of a number of normalized time-varying relations. Each relation is characterized by a fixed number of attributes and contains an arbitrary number of unique tuples. Thus a normalized relation can be viewed as a two-dimensional table in which the attributes are the columns and the tuples are the rows. Figure 1 shows a simple database that contains three relations, the supplier, the parts, and the supplier-parts relations. These relations contain information about suppliers, parts, and the available quantity from each part-supplier, respectively.

An important class of operations that manipulate the relational database is referred to as the relational algebra operations.[1] The most used operations of this class are the selection, projection, and join operations. The selection or projection and the join operations reference one and two relations, respectively. The selection operation selects from the referenced relation those tuples that satisfy a simple predicate. For example (see Figure 1), "select supplier where ADDRESS = LONDON" selects all the tuples from rela-

SUPPLIER

S#	SNAME	ADDRESS
20	SMITH	LONDON

PARTS

P#	PNAME	COLOR	WEIGHT
26	BOILER	WHITE	200

SUPPLIER-PARTS

S#	P#	QUANTITY-AVAILABLE
20	26	50

Figure 1—A simple relational database

tion supplier that contain the word "LONDON" in their address field.

Projecting a relation on a set of attributes involves first eliminating the relation's attributes (columns) that fall outside the latter set and then eliminating any duplicate tuples that may have been introduced by the first step. For example, "Project supplier on SNAME" eliminates the S# and ADDRESS columns from the supplier relation (refer to Figure 1) and removes any duplicate name in the column SNAME to produce a list (relation) of all supplier names. Joining two relations R1 and R2 on attribute (column) A of R1 and attribute B of R2 results in a new relation. A tuple in the new relation is formed by concatenating a tuple from R1 together with a tuple from R2 whenever the values of the attributes A and B satisfy a simple predicate. For example, "Join (supplier, supplier-parts) where supplier.S# = supplier-parts.S#" results in a relation with the attributes (columns) S#, SNAME, ADDRESS, P# and QUANTITY-AVAILABLE. A tuple in this relation contains, among others, the name and address of a supplier who supplies a particular part.

Another important class of operations defined for the relational databases is the update operations. These operations are the insertion, deletion, and modification.[12] The insertion (deletion) operation adds (removes) some tuples to (from) a relation in the database. The modification operation modifies the content of those tuples of a relation that satisfy a simple predicate.

The workload of a relational database system consists mainly of queries. A query is a high-level nonprocedural specification of data to be retrieved from the database. In most relational systems, the execution of a query begins by translating the query into a tree structure referred to as the query execution tree. Each node in this tree is a relational algebra operation. The edge(s) that terminates into a node represents the input relation(s) to the corresponding operation. The edge that originates from a node represents the relation which results from executing the operation represented by that node. The leaf nodes of an execution tree reference the relations in the database. Thus, the execution tree for a query defines the set of relational operations together with their input relation(s) and order of execution that results in retrieving the data specified by that particular query from the database. In most queries, the leaf nodes of the corresponding execution tree are of the selection type. Figure 2 shows a typical execution tree. Executing this tree results in a list of all supplier names that supply part #36 and located in LONDON.

3. A CLASSIFICATION SCHEME FOR DBMs

The DBMs proposed so far can be viewed as points in a three dimensional space known as the DBM space. The coordinates of this space, as indicated in Figure 3, are the indexing level, the query processing place, and the processing multiplicity.

The most fundamental and important operations the DBMs are designed to support are the selection and the modification operations. In the early designs of DBM, the latter operations were carried out using an entirely associative approach, that is, the whole database (a set of relations) was scanned and the data items that satisfy the selection (modification) criteria were retrieved (modified). Researchers in the DBMs field soon realized that such an approach was not cost-effective since it required that the whole database be scanned, at least once, for every selection or modification operation regardless of the size of the operation's result relation. To achieve a more cost-effective DBM design, a new approach for performing the selection and the modification operations has been followed. It is called the quasi-associative approach. In this approach only a relatively small portion of the database (rather than all) must be processed for every operation. To support such an approach, the database is divided into a set of data units. In order to perform the selection operation, for example, the machine must first map the corresponding selection criteria to a set of data units. These units contain, in addition to some other unwanted data, the data which satisfy the selection criteria. The data units of the latter set are then searched and the data items that satisfy the selection criteria are extracted.

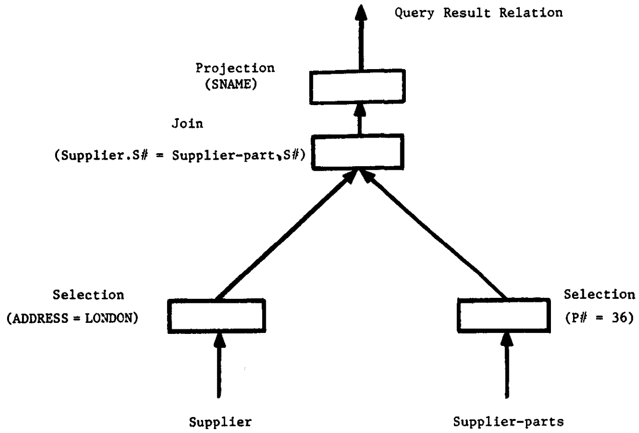


Figure 2—A typical query execution tree

The structure used in most DBMs to map the selection (modification) criteria to the relevant data units of the data-

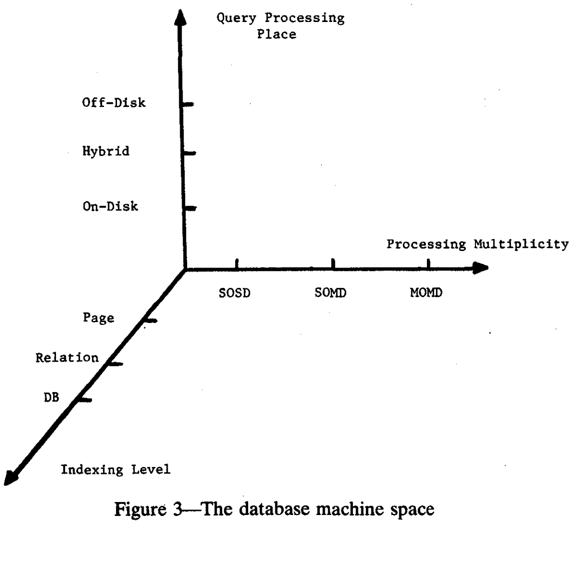


Figure 3—The database machine space

base is the index tables.[13] These tables are defined for the relations of the database and need to be stored and maintained. The data unit, the smallest addressable unit of data, can be the database (no indexing), a relation, or a fixed size page. The page size can be that of a set of tracks, a track or part of a track of a fixed/moving-head-disk.

The first coordinate in the DBM space is the indexing level defined for the database and supported by the particular DBM. Along this coordinate, the DBMs can be grouped into three categories, namely, the DBMs with database indexing level (DBMs-DB), DBMs with relation indexing level (DBMs-relation), and DBMs with page indexing level (DBMs-page). The first category includes all the DBMs that support only the associative search approach. The DBMs of both the second and third categories support the quasi-associative search approach. The index tables of a machine from the DBMs-relation category are defined for the relation as the minimum addressable unit.* In addition to supporting relation level index tables, a machine from the DBMs-page category supports index tables defined for the pages as the minimum addressable units.

The second coordinate in the proposed scheme is the query processing place. Along this coordinate, the DBMs can be grouped into three categories, namely, the off-disk,** the on-disk, and the hybrid categories. The DBMs of the first category execute the user query (and the update operations) away from the disk that stores the database. When doing that, a DBM of this category moves the data units, relevant to the query, from the disk to a separate level of processors and memories where the query processing takes place.

The DBMs of the second category execute the user query (and the update operations) on the disk where the database is stored. These machines do not need to move the data units from the disk to a different memory for processing. The disk is itself equipped with logic units that have the capability of processing the queries directly against the stored data in the disk. The DBMs of the third category execute part of the query; the selection operations (if they form the leaf nodes of the query execution tree) on the disk, move the resulting relations to a separate level of memory and processors where the rest of the query execution (if any is remaining) takes place. This category of DBMs execute the update operations directly on the disk.

The third coordinate, in the proposed scheme, is the processing multiplicity. For the on-disk (off-disk) machines, this coordinate characterizes the way the on-disk (off-disk) processing level executes the database operations.† For the hybrid machines, this coordinate characterizes the way both the on-disk and the off-disk levels execute the database operations. Along this coordinate, the DBMs can be grouped into three categories, namely, single operation stream-single data stream (SOSD),†† single operation stream-multiple data stream (SOMD), and multiple operation stream-multiple data stream (MOMD) categories. The execution of database operations on a DBM of the first or second group are done serially. That is, one operation (possibly two for the hybrid machines) is the maximum number of operations that a DBM of one of these groups is executing at any given time. While the database operation is evaluated on one tuple at a time by an SOSD machine, it is evaluated on a number of tuples simultaneously by an SOMD machine. An MOMD machine, on the other hand, executes one or more database operations (from the same query or from different queries), simultaneously. Furthermore, the operation itself is evaluated on a number of tuples, simultaneously.

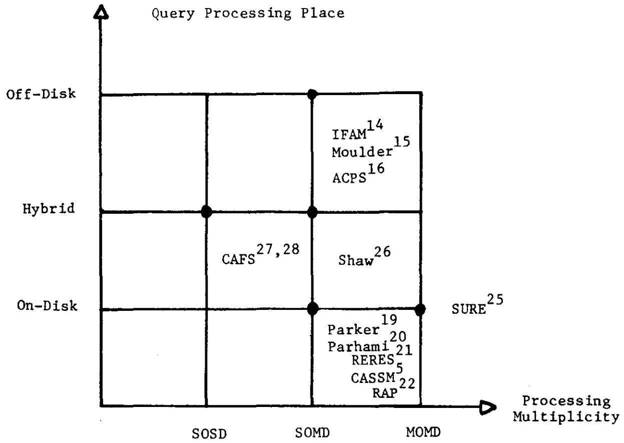


Figure 4—The DBMs with DB indexing level

4. OVERVIEW OF THE DATABASE MACHINES

The Database Machines With Database Indexing Level (DBMs-DB)

Figure 4 presents the DBMs that fall within the DBMs-DB category. Historically, the members of this group (except the hybrid ones) are the first DBMs to be proposed. An off-disk machine of this group (refer to it as off-disk-DB machine)[14,15,16] is generally organized, as shown in Figure 5, around a "bit/byte serial-word parallel" associative memory (AM).[17] This type of AM is organized as a set of relatively short memory words. Each word is implemented as a high speed serial shift semiconductor register and is associated with a processing element (simple hardwired logic unit with one bit/byte data path). The processing elements under the direction of a single hardwired control unit, execute parallel search, parallel updates, and simple arithmetic operations on

* To facilitate parallel processing and data movement, some DBMs of the first/second category store the corresponding minimum addressable unit of data (DB/relation) on a set of physical units, the minimum access units (MACUs). Each can be moved separately. When a data item needs to be retrieved, however, all the MACUs containing the DB/relation are processed. In the DBMs of the third category, the page (the minimum addressable unit) is contained in one MACU.

** The disk here implies a moving-head-disk, a fixed-head-disk, or an electronic disk such as the magnetic bubble memory (MBM) or the charge coupled devices memory (CCD). The disk is the unit of the mass storage memory that stores the database.

† By database operations we mean the relational algebra operations, selection projection and join, and the update operations.

†† Within the context of relational databases, we define the data stream as a sequence of nondecomposable data items, with the tuple being that item.

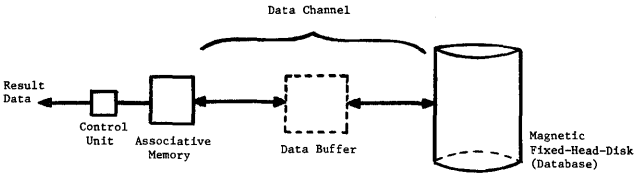


Figure 5—A general architecture for the off-disk-DB machines

the content of the memory words. The database is stored on a magnetic fixed-head-disk (head-per-track disk)[18] and is provided with a data channel to load/unload the AM from/to this disk. In some implementations, this channel includes a high speed semiconductor buffer to facilitate the loading and unloading of the AM, and also permits the content of a small number of disk tracks to be transferred, simultaneously.

The off-disk-DB machines implement only the selection and update operations. To perform the selection operation, for example, the database is divided into units each with a capacity equal to that of AM. These units are sequentially loaded into AM where the content of each data unit is searched in parallel to identify those tuples that satisfy the selection criteria.

The off-disk-DB machines are inadequate for many reasons. The most important ones are the high cost of such organization especially when supporting large and very large databases, and the AM load/unload bottleneck that renders the whole organization useless. The high cost of an off-disk-DB machine is attributed to the high cost of its two components, the AM and the fixed-head-disk. Although the cost of semiconductor memory and logic have dropped substantially in the past few years, the cost of impending logic in semiconductor memories on a massive scale as required by large AMs continues to be very high. The use of a magnetic fixed-head-disk to store large volumes of data is cost-ineffective compared with other mass storage devices such as the moving-head-disk. The latter disk provides a storage capacity one order of magnitude higher than that of the fixed-head-disk and a moderate decrease in its bandwidth. The magnetic fixed-head-disk is rapidly becoming an obsolete technology for mass storage.

Because of the limitation on the size of AM, the implementation of the selection or update operations requires that the database be divided in data units, each having the capacity of AM. To process a database operation, these units must be loaded, one at a time, from the disk into the AM where the parallel search/update takes place. The AM is a very fast search/update device once the data unit is loaded into it, but the loading of a data unit from the slow rotating mechanical fixed-head-disk requires a substantial amount of time. Thus the data movement creates a system bottleneck and limits the speed of executing the database operations to that of the slow disk rather than the AM. To overcome the problem, the fixed-head-disk as well as the channel connecting such a disk to AM must be modified to allow many disk tracks to be transferred in parallel. Such a solution, however, increases the cost of such organization. Thus we can conclude that the data movement problem causes this organization to be cost-ineffective in supporting the selection and update operations.

Most of the on-disk-DB machines have been organized as SOMD machines. A machine of the latter groups[5,19,20,21,22] is generally organized, as shown in Figure 6, around a logic-per-track (LPT) disk.[23] The LPT disk which stores the database is a head-per-track (fixed-head) disk in which each head (track) is associated with a logic unit. This unit is provided with a very limited amount of memory and has the capability of processing the data of the associated track "on-the-fly." That is, the logic unit can search and/or update the data of the associated track as it passes under the corresponding head. The set of logic units associated with an LPT disk is controlled and directed by a master control unit (refer to Figure 6) to carry out a parallel search or update on the database stored on the disk. Some of the LPT machines use the parallel search and update capabilities of the LPT disk to implement only the selection and update operations.[19,20,21] Others[5,22] use such capabilities to implement the various relational system operations, namely, the selection, projection, join, and update operations. The execution of the selection operation on an LPT machine takes normally one disk revolution. During this revolution, the disk logic units search the data stored on the corresponding tracks in parallel and retrieve those tuples that satisfy the selection criteria. On the other hand, the execution of the join or projection operation requires many disk revolutions. The number of revolutions required for the join operation, for example, is on the order of the number of distinguished values in the join attribute of the smaller relation participating in the join operation.

The LPT DBM organization has several important advantages. Among these are the speed by which the selection and update operations are executed and the elimination of the data movement problem that faces the off-disk-DB machines. The fact that these operations are executed directly by logic units attached to the track heads of the mass storage device where the database resides causes only the data resulting from these operations (usually a very small fraction of the database) to be moved out from the disk rather than all the databases. Despite the stated advantages, the LPT DBM organization has a number of serious drawbacks. The most important one is the high cost attached to this organization. This is attributed to the cost of the two basic components of the LPT disk, namely the fixed-head-disk and the vast amount of logic associated with it. This combination increases the cost of

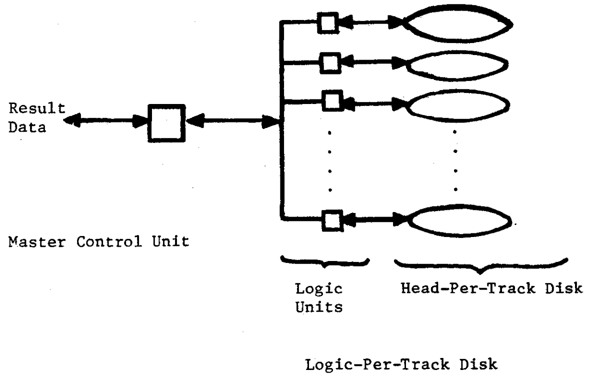


Figure 6—A general architecture for the logic-per-track database machines

the LPT disk to higher levels than the more traditional mass storage device, the moving-head-disk, and severely limits the size of the database that can be supported by the DBMs of the LPT type. Another important problem that faces the LPT DBM is that such a machine is not suitable for processing the projection or the join operation. This is because the processing of such operations requires many disk revolutions, and the disk is a slow mechanical device that needs a relatively large amount of time to complete one revolution. The poor performance of the LPT machines in supporting the projection and join operations is evident in a performance study carried out for the LPT machine RAP.[24] This study shows that a RAP-like machine has no speed advantage over that of a conventional uniprocessor relational system when executing the projection or joirl operation.

SURE[25] is another on-disk-DB machine, designed primarily to execute the selection operation. It is organized, as shown in Figure 7, around a moving-head-disk modified for parallel reading/writing from/to all the tracks of one disk cylinder. The parallel data streams from the disk are joined together and fed to a number of parallel search units. These units execute different selection operations on the combined data streams. Thus, such a machine is a MOMD machine. The execution of a selection operation requires that the joined data streams from all the disk cylinders be searched on-the-fly by the corresponding search unit. The advantages of such organization are its relatively low cost in comparison with the previously presented machines, and its ability to handle a number of selection operations in parallel. However, this machine is an expensive improvement for its performance, since a selection operation on SURE requires as many disk revolutions as the number of cylinders on the disk.

Shaw's machine[26] and CAFS (*C*ontent *A*ddressable *F*ile *S*tore)[10,27,28] have been organized as hybrid-DB machines. The former, as shown in Figure 8, is an SOMD machine which combines two levels of memory and logic. The first level employs the LPT disk as a mass storage device to store the database and execute the selection operations (if they form the leaf nodes of the corresponding query execution tree) and the update operations. The second level employs an associative memory to execute the rest of the query when any is remaining. Both levels are connected via a channel for data-movement. The combination of two structures in Shaw's machine each of which is tailored to the processing of a class of database operations, substantially improves its performance. It also eliminates the data movement problem and continues to have both the selection and the more complex operations (join and projection) carried out with high performance. This machine, however, has a compounded cost problem since it combines two very expensive technologies, namely the LPT disks and the associative memories, both causing it to be unsuitable to support large and very large databases.

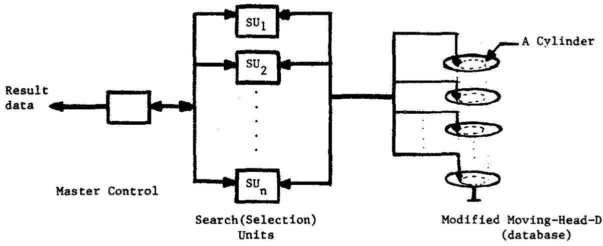


Figure 7—SURE database machine organization

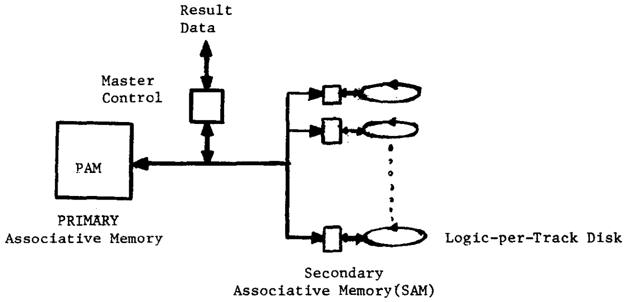


Figure 8—Shaw's hybrid-DB machine organization

CAFS,[10,27,28] an SOSD hybrid-DB, combines, as shown in Figure 9, two levels of logic units. The first level is organized around a set of moving-head-disks that store the database. Each of these disk units can only readout one of its tracks at a time. The data output from all the disk units are combined into a single data stream and fed to a search unit (SU). The SU executes one selection operation at a time in on-the-fly fashion against the incoming stream of data (tuples). Those tuples that satisfy the selection operation are then sent to the machine's second level where they are projected or joined with tuples that are selected during the execution of another selection operation. The off-disk logic unit (the join/projection unit) performs the projection or join operation in linear time (in time proportional to the number of tuples to be joined or projected) using some novel algorithms.[27]

The Database Machines With Relation Indexing Level (DBMs-Relation)

Figure 10 presents those DBMs that fall within the DBMs-Relation category. A DBM within this group defines, stores, and maintains an index table for the relations in the database.

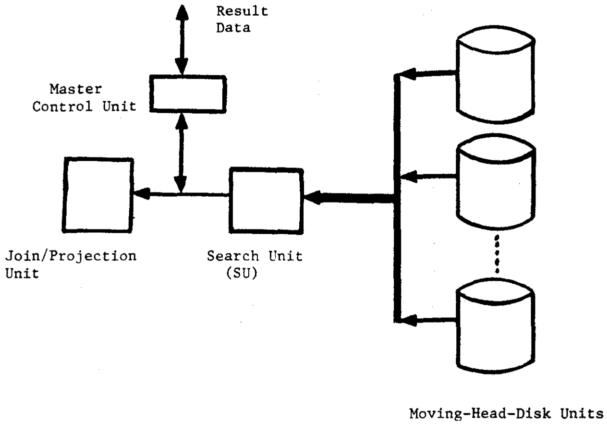


Figure 9—CAFS database machine organization

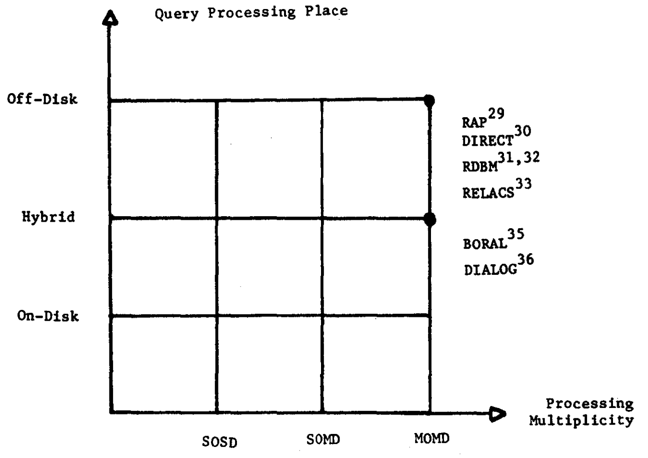


Figure 10—The DBMs with relation indexing level

For each of these relations, the table stores the physical addresses of all the tracks of the mass storage disk units that store such relations.

In general, the DBMs-Relation DBMs are organized, as shown in Figure 10, as MOMD machines. These machines are clustered into the off-disk and hybrid groups. The off-disk group includes the DBMs RAP.2,[29] DIRECT,[30] RDBM,[31,32] and RELACS.[33] A general configuration for an off-disk-relation machine is shown in Figure 11. The database is stored on a set of moving-head-disk units that are connected to an off-disk processing level through a data channel. In RAP.2, the off-disk processing level is organized as an LPT disk. The disk uses the magnetic bubble memory (MBM)[34] technology instead of the traditional magnetic disk technology used in RAP design as the storage medium. RAP.2 implements all the database operations using the parallel search and update capabilities of the LPT disk. To execute the selection operation on RAP.2, for example, the relation referenced by such operation must first be brought in total from the moving-head-disk units into the LPT disk if it is not already there. The logic units, which correspond to the tracks of the LPT disk that store the buffered relation, search the tracks in parallel and on-the-fly and retrieve those tuples that satisfy the selection criteria. To execute the join operation, both of the referenced relations must be brought into the LPT disk. RAP.2 can also execute a number of database operations simultaneously (MOMD machine) provided that the relations referenced by these operations can be stored, in total, in the LPT disk.

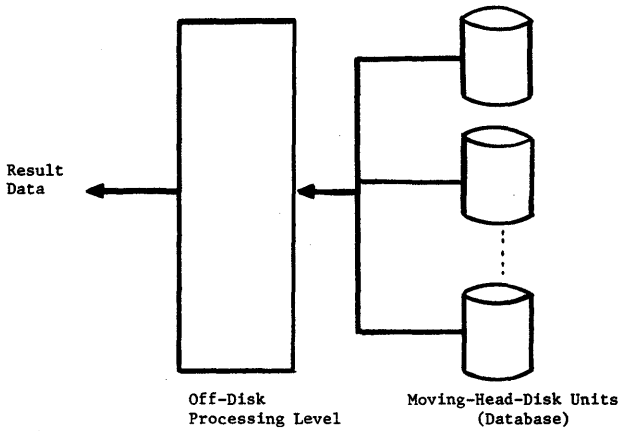


Figure 11—A general organization for the off-disk-relation DBMs

RAP.2 overcomes many of the problems associated with its predecessor RAP. Some of these problems are the limitation on the size of the database and the number of simultaneous operations that can be supported by RAP. RAP.2 uses moving-head-disk units (rather than the LPT disk) to store the database, thus substantially increasing the size of the stored database. Furthermore, RAP.2 is provided with the ability to process a number of operations, simultaneously. Despite all of the improvements, RAP.2 continues to have many problems, namely, the limitation of the size of any relation in the database to that of the LPT disk and the bottleneck created as a result of the great difference between the search speed of the LPT disk and the speed by which this disk is loaded/unloaded from/to the moving-head-disk units.

The off-disk processing level of DIRECT[30] is organized as shown in Figure 12. The off-disk memory, implemented using the charge coupled device (CCD) technology, is partitioned into a number of blocks (or, in DIRECT terminology, frames), each with the capacity of a track. The memory is used as a cache to store all or part of a relation(s) while it is being processed by the logic units. The logic units in DIRECT are microprocessors, each with its own random-access-memory (RAM). This memory is large enough to store a number of the cache-memory blocks, simultaneously. A logic unit in DIRECT is not associated with a specific memory block (as in the LPT disk). Instead, all the logic units are interfaced to the memory through a crossbar switch, thus sharing all of its blocks.

DIRECT implements all of the database operations using the multiprocessing capability of the off-disk processing level. The execution of these operations are not carried out in an on-the-fly fashion. A logic unit executing a given operation reads first part of the referenced relation(s) from the cache frames (if the referenced data is in cache, otherwise it has to be brought in from disk) into its internal memory and then the operation is executed against the content of this memory.

Since DIRECT organization does not require that all the relation has to be in the cache before its processing can start, DIRECT, therefore, can support the processing of a relation of any size. Furthermore, the multiprocessing organization of DIRECT lends itself better than that of RAP.2 to support the simultaneous execution of a large number of operations. By providing each logic unit with a large internal memory, DIRECT eliminates the need to process data on-the-fly. De-

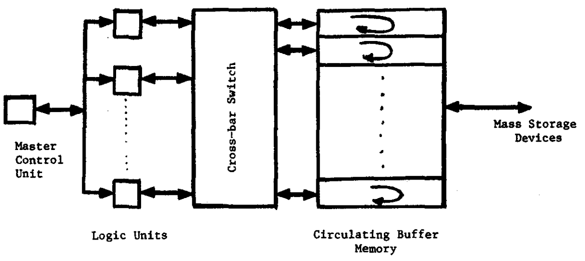


Figure 12—The organization of the off-disk processing level in DIRECT

spite all of these improvements, DIRECT organization, on the other hand, has several problems. The most important one is the disk/cache bottleneck resulting from the long time required to load the cache with relations from the slow mechanical moving-head-disk. Another important problem is the long time required to manage the simultaneous execution of a number of operations and the cache memory. This time dominates the execution time of almost any database operation executed by DIRECT.

The off-disk processing level of RELACS[33] is organized as shown in Figure 13. This level includes two associative memory units (AM_0 and AM_1), a buffer memory, and a high speed channel to load the AM units from this buffer. The buffer is partitioned into a set of blocks each having the same capacity as that of one AM. The block is implemented as a high speed semiconductor shift registers. RELACS implements the database operations using the parallel search and update capabilities of the two AM units. These units execute two different operations, simultaneously. To execute the selection operation, for example, the relation referenced by such operation is first loaded into successive blocks of the buffer memory. As one block is being loaded from the disk units, another block is read into one AM unit to be searched in parallel for those tuples that satisfy the selection criteria. This is repeated for all the blocks of the referenced relation.

RELACS overcomes many of the problems that are associated with its predecessor, an off-disk-DB machine. It overcomes both the AM load/unload problem as well as the limited size of the database that can be supported by such machine. Despite these improvements, RELACS continues to be costly because of the use of unconventional associative memory technology. Furthermore, RELACS has an inflexible structure that can support the simultaneous execution of only two operations.

The group of hybrid DBMs that support relation level index includes Boral's DBM[35] and DIALOG[36] as shown in Figure 10. The former machine stores the database on modified moving-head-disk units. Each one of these units is similar to the one used by the SURE machine in Figure 7. On the other hand, DIALOG stores the database on head-per-track disk units implemented using magnetic bubble memory technology.[34] In Boral's machine, every head in a modified moving-head-disk unit is provided with a logic unit of its own. Under the direction of a control unit, the set of logic units associated with a disk can process, on-the-fly, the data stored on one disk cylinder at a time. Such capability is used by Boral's machine to implement the selection and update operations.

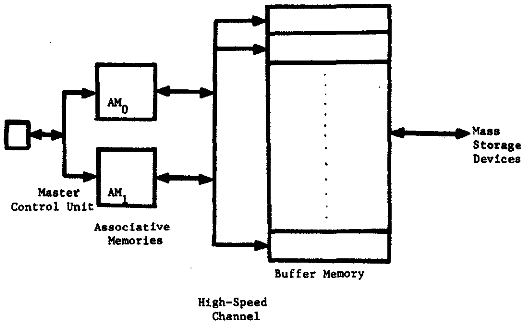


Figure 13—The organization of the off-disk processing level in RELACS

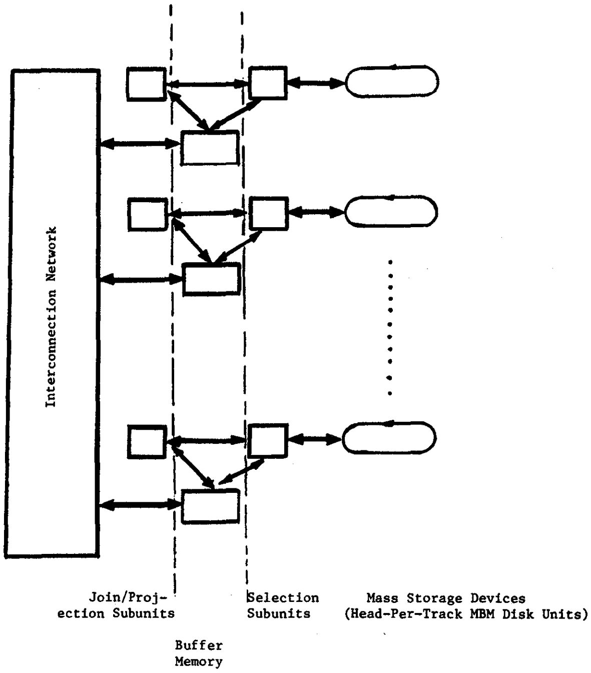


Figure 14—The organization of DIALOG database machine

The off-disk processing level of Boral's machine consists of a set of microcomputers, each with its own internal RAM memory and small secondary disk unit. The set of microcomputers implement the projection and join operations using algorithms that employ parallel sorting techniques.[37] Boral's machine uses a bus similar to Ethernet's to interconnect the elements of the on-disk and off-disk processing levels.

DIALOG,[36] as shown in Figure 14, provides each head of its head-per-track disk units with its own processing unit. An individual unit consists of selection, join/projection, and buffer memory subunits. The selection subunit is connected directly to the head of a disk track and implements the selection and update operations by processing the track's data on-the-fly. Furthermore, the selection subunit is interconnected to the join/projection subunit either directly or through the buffer memory. The buffer subunits of all DIALOG's processing units are interconnected via an interconnection network. This network is essential for implementing the join and projection operations on relations which span multiple disk tracks. In Figure 14, it can be seen that the selection subunits form the on-disk processing level of DIALOG. Conversely, the join/projection subunits, the buffer memories, and the interconnection network form the off-disk processing level of DIALOG.

The Database Machines With Page Indexing Level (DBMs-Page)

Figure 15 presents those DBMs that fall within the DBMs-Page category. A DBM of this type defines, stores and main-

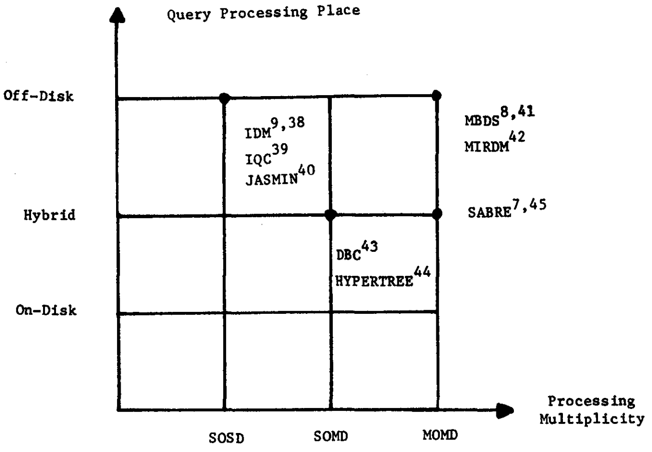


Figure 15—The DBMs with page indexing level

tains two types of index tables, namely the relation level index and the page level indexes. The former index table is similar to the one defined for a DBM of the DBMs-Relation type. The page level index tables are defined on the set of the most frequently referenced attributes of the database. For every value of each of these attributes, the corresponding index table stores the set of addresses of all the pages which contain tuples having such a value.

Two groups of machines can be distinguished within the DBMs-Page category. They are the off-disk and the hybrid groups (refer to Figure 15). The off-disk SOSD group includes the DBMs IDM,[9,38] IQC,[39] and JASMIN.[40] A DBM machine of this type stores the database on a set of moving-head-disks and has relatively small size pages, the size of a track. The off-disk processing level is a single von Neumann processor interfaced to the mass storage devices through a large RAM memory (Main memory). The processor is dedicated to database processing and implements the various database operations. This implementation depends heavily on the page level index tables to speed up the execution of these operations. To perform the selection operation, for example, the corresponding selection criteria is first processed against the page level index table to obtain a set of page addresses. Each page in this set contains at least one tuple that satisfies the selection criteria. The set of pages is then brought into the main memory where it is searched by the processor for those tuples that satisfy the selection criteria.

The off-disk MOMD group includes the machines MBDS[8,41] and MIRDM[42] shown in Figure 15. The former machine is organized as a collection of microcomputers, each with its own main RAM memory and moving-head-disk units. The set of microcomputers is interconnected through a time-shared bus and controlled by another microcomputer. Each microcomputer maintains page level index tables defined for the portion of the database that is stored on its own mass storage devices. The page size in MBDS is relatively small. With the aid of these tables, a microcomputer can execute any database operation provided that the referenced pages reside on its mass storage devices. When the referenced pages span several microcomputers, then these microcomputers must cooperate to execute the corresponding database operation.

MIRDM[42] is another machine organized as off-disk MOMD and provided with page level index tables. This machine stores the database on modified moving-head-disk units similar to the ones used by SURE. The off-disk memory, Figure 16, is implemented using the charged coupled device or random-access-memory technology. The memory is partitioned into a number of blocks, each with the capacity of a disk cylinder (the page size in MIRDM). It is used as a cache to store some pages while being processed by the logic units which in MIRDM are microprocessors, each with its own RAM memory. The microprocessors are organized in clusters that are interfaced to the buffer memory through a cross-bar switch. Different clusters can execute different database operations simultaneously or cooperate to execute one database operation in parallel.

The group of the hybrid DBMs that support page level index includes DBC,[43] HYPERTREE[44] and SABRE shown in Figure 15.[7,45] Each of the first two machines is organized as an SOMD machine, whereas SABRE is organized as an MOMD machine. The DBC stores the database on a set of modified moving-head-disk units similar to those of MIRDM. The page level index in DBC is defined for pages having the capacity of one disk cylinder. The on-disk processing level of this machine consists of a set of logic units equal in number to that of the tracks in one disk cylinder. The set of logic units are interfaced to the mass storage devices through a switch that permits only one disk at a time to be connected to these units. The logic units have the capability to process, on-the-fly, the data stored on a cylinder of the disk selected by the current setting of this switch. The on-disk processing level in DBC uses the parallel search and update capabilities of the logic units together with the page level index tables to implement the selections and the update operations.

The off-disk processing level of DBC consists of a set of microprocessors, each with its own memory and controlled by another microprocessor. This level of processors implement the join and the projection operations.

The HYPERTREE[44] DBM consists of a set of processors interconnected together to form a special type of tree, called the hypertree. This tree is essentially a binary tree in which each node (processor) is normally connected to one of its

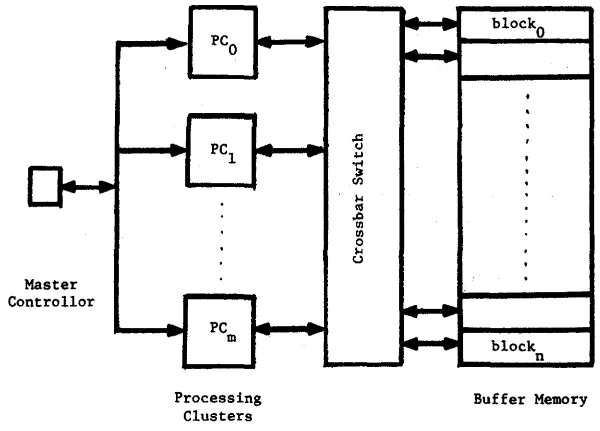


Figure 16—The off-disk level organization of MIRDM

siblings. The leaf nodes are interconnected using the perfect shuffle structure.[46] Furthermore, each of the leaf nodes (processors) is connected to a moving-head-disk. The leaf processors are responsible for executing the selection and update operations, and the higher order processors are responsible for executing the join and the projection operations.

5. A CRITICAL LOOK AT THE PREVIOUSLY PROPOSED DBMs

Historically, the first DBMs to be proposed were organized as Off-Disk machines and were provided with only the associative access to the database (Off-Disk-DB DBMs). In general, these machines (and this design approach) had many drawbacks particularly their ineffectiveness in handling the very large databases. A DBM of this type has to move all the databases from the slow rotating mechanical disks where it is stored, to a very fast (associative) memory where the execution of the selection, the update, or the other database operations would take place. In the large database environment, the I/O channels in these machines easily become a system bottleneck as a result of large differences between the speed of loading the associative memory and the speed of executing the search/update operation. This drawback, coupled with the high cost of producing associative memory units of large capacities, storing the database on head-per-track disks, and proving the DBM with very wide I/O channels, make this design approach cost-ineffective.

The On-Disk-DB design eliminates the data movement problem by processing the database operations on the mass storage device where the database itself resides, and only the result of these operations are moved out of the devices. Although this approach eliminates the need to move large volumes of data and eliminates the I/O bottleneck, it continues to have many drawbacks. The most important one is the "on-the-fly" processing problem, i.e., the need for a hardware unit to process the data as fast as the disk head delivers it. A logic unit for on-the-fly processing is difficult to design because of the variations in the data processing requirement. The solution is to associate a RAM buffer with size equal to that of a disk track with each logic unit. With today's RAM technology, this should be feasible for disks with a small number of heads. It is not feasible, however, for a logic-per-track disk because a RAM memory with capacity equal to that of the LPT disk itself is needed. It is interesting to note that the on-the-fly processing problem is not unique to the on-disk-DB machines, but it is shared by machines of all other categories that use such techniques to carry out one or more of the database operations.

A query can be thought of as a tree whose nodes are database operations. The leaves of the tree reference relations from the database. In a real database environment the leaf nodes are mostly of the selection and update types. A hybrid DBM processes the leaf selection operations and, in some machines, the update operations in the disk. The relations resulting from this processing (referred to as the temporary relations) are then moved to a fast processor-memory complex where the rest of the query operations (if any) are executed. In most cases executing the leaf selection and update operations in the disk largely reduces the volume of data needed to be moved to the fast processor-memory complex.

From the above, one can conclude that the performance of the DBMs of the Hybrid-DB design approach is superior to that of both the On-Disk-DB and the Off-Disk-DB DBMs. This is because the DBMs of the former type execute the database operations on more tailored hardware units and reduce the volume of data to be moved out of the mass storage disks at the same time.

In general, the number of tuples which are selected (modified) as a result of executing a typical selection (modification) operation is a very small fraction of those in the database. In the light of the current processor memory technology and the vast amount of data in the very large databases, it is clear that the need to scan the whole database at least once and to extract (update) this very small fraction of the database as required by a DBM of DBMs-DB type is very cost-ineffective. Actually, the need to scan the whole database resulted in DBMs with very expensive parallel search components such as LPT disk units and associative memories. The design approach, which provides the DBM with a mechanism that eliminates the need to scan all the database for each data operation is highly desirable.

In the context of the DBMs, an index table defined on the database has been used as a mechanism to reduce the amount of data to be processed for a given selection or modification operation. For every relation in the database, the index table of a DBM which belongs to the DBMs-Relation category stores the set of addresses of the disk tracks that store the corresponding relation. Although the size of such an index table (relative to that of the database) is a function of the number of relations in the database and the track size, it is, nevertheless, very small. Its maintenance, storage cost, and access time are negligible relative to that of the database. To execute a selection (modification) operation on an On-Disk-Relation/Hybrid-Relation DBM, only the data units which store the relation referenced by the operation are searched (modified). For those machines which use a logic-per-track disk as a storage media, only the tracks which correspond to these units are processed. The rest of the disk tracks can be processed simultaneously and in parallel with one or more other selection and/or update operations. On the other hand, to execute these operations on an Off-Disk-Relation DBM, only the data units which store the relation referenced by the operation need be moved to the processor-memory complex for processing. This suggests that the DBMs which support only a relation level indexing perform differently under different types of databases.

In the context of databases dominated by relatively small size relations, indexing on the relation level substantially improves the DBM cost-effectiveness relative to that of the no indexing ones. For the on-disk or hybrid organized DBMs, the logic-per-track disk can be replaced by a less expensive mass storage media. For the Off-Disk DBMs, the amount of data which needs to be moved out of the secondary store is substantially reduced. Thus, a less expensive I/O channel can be utilized. Conversely, in the context of the databases dominated by relatively large size relations (in the very large databases, a relation could have the size of several magnetic disks), indexing on the relation level does not improve the DBM cost-effectiveness. It is very clear that such a DBM is hampered by the same problems as that of the DBMs-DB type.

The index tables supported by the DBMs which belong to the DBMs-Page category are defined on the set of the most frequently referenced attributes of the database. For every value of each of these attributes, the corresponding index table stores the set of addresses of all the pages which contain tuples having such value. Although the size of this index (relative to that of the database) is a function of the number of attributes in the database, the number of distinguished values of each attribute, and the size of the page itself, nevertheless, the index size and its storage maintenance cost is substantial. In conjunction with the page index table, a clustering mechanism has to be used.[43] This mechanism is used to store the set of tuples which are frequently referenced together in as few pages as possible.

In general, the use of the page indexing coupled with an efficient clustering mechanism improves the performance of the selection and possibly the modification operations. The need to search or update the page index when executing the selection or update operation introduces some delays in the processing of queries. In the case of the very large database systems, the use of small size pages results in a relatively large page index table which requires a huge amount of storage, large access time, and high maintenance and updating cost. It enables designers, however, to use the traditional moving-head disk units as mass storage devices as seen from the review of DBMs which support page level index tables with small page sizes. The use of large size pages in conjunction with a DBM results in a relatively small page index (~1% of the total database size[43]) and its storage cost, access time, and maintenance cost are substantially lower. The performance of the selection and the modification operations are greatly enhanced especially if a page is processed by a number of processors in parallel. One drawback to this approach is the need to use modified moving-head-disk units that cost much more than the traditional moving-head-disk.

In the scheme presented earlier, the previously proposed DBMs have been organized as SOSD, SOMD and MOMD machines. Because of the relatively low cost of the processor and memory devices, the use of parallel processing enhances the effectiveness of the DBMs. Although the SOMD organization of the DBMs reduces the execution time of a database operation, it does not offer a real solution to the database multiple-user problem. The MOMD organization is more effective in supporting databases where fast parallel accesses to the databases is a basic requirement. This is because the MOMD organization has the ability not only to execute a database operation in parallel but also to execute more than one operation from varying queries simultaneously and in parallel.

One important drawback in the MOMD organization is the associated overhead, namely, the large amount of processing needed for controlling the simultaneous execution of the different operations and the management of the various system components. In most MOMD DBMs such an overhead puts an upper limit on the number of queries and resources that can be simultaneously active in the system. Minimizing such overhead, therefore, must be a basic objective for the MOMD DBM designer.

REFERENCES

1. Codd, E. F. "A Relational Model for Large Shared Data Banks." *Communications of ACM,* (1970), pp. 377–387.
2. Codd, E. F. "Further Normalization of the Data Base Relational Model." *Data Base Systems* (Courant Computer Science Symposia, Vol. 6). Englewood Cliffs, N.J.: Prentice-Hall, May 1971.
3. Kim, W. "Relational Database Systems." *ACM Computing Surveys,* 11 (1979), pp. 185–211.
4. Ozkarahan, E. A., S. A. Schuster and K. C. Smith. "RAP—An Associative Processor for Database Management." *AFIPS, Proceedings of the National Computer Conference* (Vol. 45), 1975, pp. 379–387.
5. Lipovski, G. J. "Architectural Features of CASSM: A Context Segment Sequential Memory." *Proceedings of the Fifth Annual Symposium on Computer Architecture,* Palo Alto, California, April 1978, pp. 31–38.
6. Boral, H., D. J. Dewitt, D. Friedland, N. Jarrel, and W. K. Wilkinson. "Implementation of the Database Machine DIRECT." *IEEE Transactions on Software Engineering,* 8 (1982), pp. 533–543.
7. Gardarin, G. "An Introduction to SABRE: A Multi-Microprocessor Database Machine." *The Sixth Workshop on Computer Architecture for Non-Numeric Processing,* Hyeres, France, June 1981.
8. Hsiao, D. K., D. S. Kerr, A. Orooji, Z-Z. Shi, and P. P. Strawser. "The Implementation of a Multibackend Database System (MDBS): Part I—An Exercise in Database Software Engineering." In *Advanced Database Machine Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, 1983, pp. 300–326.
9. Britton-Lee, Inc. "IDM500—Intelligent Database Machine." Product Description, Britton-Lee, Inc., Los Gatos, California, USA, 1980.
10. CAFS-800 Product Description. ICL Headquarters, Putney, London SW15 ISW, England, 1979.
11. Canaday, R. H. "A Backend Computer for Database Management." *Communications of the ACM,* (1974), pp. 575–582.
12. Ullman, J. D. *Principles of Database Systems.* Rockville, Md.: Computer Science Press, 1980.
13. Martin, J. *Computer Data-Base Organization* (2nd ed.) Englewood Cliffs, N.J.: Prentice-Hall, 1977.
14. Defiore, C., and P. B. Berra. "A Data Management System Utilizing an Associative Memory." *AFIPS, Proceedings of the National Computer Conference* (Vol. 42), 1973, pp. 181–185.
15. Moulder, R. "An Implementation of a Data Management System on Associative Processor." *AFIPS, Proceedings of the National Computer Conference* (Vol. 42), 1973, pp. 171–179.
16. Linde, R., R. Gates, and T. Peng. "Associative Processor Application to Real-time Data Management." *AFIPS, Proceedings of the National Computer Conference* (vol. 42), 1973, pp. 187–195.
17. Yau, S. S., and H.S. Fung. "Associative Processor Architecture—A Survey." *ACM Computing Surveys,* 9 (1977), pp. 3–27.
18. Matick, R. *Computer Storage Systems and Technology.* New York: Wiley-Interscience, 1977.
19. Parker, J. L. "A Logic per Track Retrieval System." *Proceedings of IFIP Congress,* 1971, pp. TA-4-146 to TA-4-150.
20. Parhami, B. "A Highly Parallel Computing System for Information Retrieval." *AFIPS, Proceedings of the National Computer Conference* (Vol. 41, Part II), 1972, pp. 681–690.
21. Lin, C. S., C. P. Smith, and J. M. Smith. "The Design of a Rotating Associative Memory for Relational Database Systems." *ACM Transactions on Database Systems,* 1 (1976), pp. 53–65.
22. Ozkarahan, E. A., S. A. Schuster, and K. C. Smith. "RAP—An Associative Processor for Database Management." *AFIPS, Proceedings of the National Computer Conference* (Vol. 45), 1975, pp. 379–387.
23. Slotnich, D. L. "Logic per Track Devices." In *Advances in Computers.* New York: Academic Press, 10 (1970), pp. 291–296.
24. Ozkarahan, S., A. Schuster, and K. C. Sevcik. "Performance Evaluation of a Relational Associative Processor." *ACM Transactions on Database Systems,* 2 (1977), pp. 175–195.

25. Leilich, H. O., G. Stiege, and H. Ch. Zeidler. "A Search Processor for Database Management Systems." *Proceedings of Conference on Very Large Databases,* 1978, pp. 280–287.
26. Shaw, D. "Knowledge-Based Retrieval on a Relational Database Machine." Ph.D. Thesis, Department of Computer Science, Stanford University, 1980.
27. Babb, E. "Implementing a Relational Database by Means of Specialized Hardware." *ACM Transactions on Database Systems,* 4 (1979), pp. 1–29.
28. Maller, V. A. "Information Retrieval Using the Content Addressable File Store." *Proceedings of the IFIP Congress,* 1980, pp. 187–192.
29. Schuster, S. A., H. B. Nguyen, E. A. Ozkanahan, and K. C. Smith. "RAP-2: An Associative Processor for Database and its Applications." *IEEE Transaction on Computers,* C-28 (1979), pp. 446–458.
30. Dewitt, D. J. "DIRECT—A Multiprocessor Organization for Supporting Relational Database Management Systems." *IEEE Transactions on Computers,* C-28 (1979), pp. 395–408.
31. Schweppe, H., H. Zeidler, W. Hell, H. Leilich, G. Stiege, and W. Teich. "RDBM—A Dedicated Multiprocessor System for Database Management." In *Advanced Database Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, 1983, pp. 36–86.
32. Auer, H., W. Hell, H. Leilich, H. Schweppe, G. Stiege, S. Seehusen, J. Lie, H. Zeidler, and W. Teich. "RDBM—A Relational Database Machine." *Information Systems,* 6 (1981), pp. 91–100.
33. Oliver, E. "RELACS, An Associative Computer Architecture to Support a Relational Data Model." Ph.D. dissertation, Syracuse University, 1979.
34. Chen, T. C., and H. Chang. "Magnetic Bubble Memory and Logic." In *Advances in Computers* (Vol. 17). New York: Academic Press, 1978.
35. Boral, H. "On the Use of Data-Flow Techniques in Database Machines." Ph.D. Thesis, Computer Sciences Department, University of Wisconsin—Madison, 1981.
36. Wah, B. W., and S. B. Yao. "DIALOG—A Distributed Processor Organization for Database Machines." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 243–252.
37. Bitton, D., H. Boral, D. Dewitt, and W. K. Wilkinson. "Parallel Algorithms for the Execution of Relational Database Operations." *ACM Transactions on Database Systems,* (1983), pp. 324–353.
38. Epstein, R., and P. Hawthorn. "Design Decisions for the Intelligent Database Machine." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 237–241.
39. Sekino, A., K. Takeuchi, T. Makino, K. Hakozaki, T. Doi, and T. Goto. "Design Consideration for an Information Query Computer." In *Advanced Database Machine Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, 1983, 130–167.
40. Fishman, D., M. Y. Lai, and W. Wilkinson. "Overview of the Jasmin Database Machine." *Proceedings of the Special Interest Group on Management of Data* (SIGMOD), June 1984, pp. 234–239.
41. He, X.-G., M. Higashida, D. S. Kerr, A. Orooji, Z.-Z. Shi, P. R. Strawser, and D. K. Hsiao. "The Implementation of a Multibackend Database System (MDBS): Part II—The Design of a Prototype MDBS." In *Advanced Database Machine Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, 1983, 327–385.
42. Qadah, G. Z., and K. B. Irani. "A Database Machine for Very Large Relational Databases." *Proceedings of the International Conference on Parallel Processing,* August 1983, pp. 307–314.
43. Banerjee, J., D. Hsiao, and K. Kannan. "DBC—A Database Computer for Very Large Databases." *IEEE Transactions on Computers,* C-28 (1979), pp. 414–429.
44. Goodman, J. R. "An Investigation of Multiprocessor Structures and Algorithms for Database Management." Memo No. UCB/ERLM81, College of Engineering, University of California/Berkeley, May 1981.
45. Gardarin, G., P. Bernadat, N. Temmerman, P. Valduriez, and Yann Viemunt. "SABRE: A Relational Database System for a Multimicroprocessor Machine." In *Advanced Database Machine Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, 1983, pp. 19–35.
46. Stone, H. S. "Parallel Processing with Perfect Shuffle." *IEEE Transactions on Computers,* C-20 (1971), pp. 153-161.

Z80,000 32-bit microprocessor

by ANIL PATEL*
Zilog, Inc.
Campbell, California

ABSTRACT

With recent advances in very large scale integrated (VLSI) circuit design, the formerly distinct boundaries between micro-, mini-, and mainframe computer architectures are eroding. For example, the Motorola 68000, the Z8000 CPU, and the Intel 8086 have broken the boundary between micro- and minicomputer. Now, the Z80,000 CPU, Zilog's new 32-bit processor chip, has broken the distinct boundary between mini- and mainframe computers by featuring a mainframe power on an integrated-circuit chip. The distinguishing features of the Z80,000 CPU—such as on-chip virtual memory management, on-chip cache memory, six-stage pipeline architecture, burst memory transfer, and multiprocessing support, put it ahead of any CPU in its class. It supports linear and segmented addressing. The regular instruction set and rich and powerful addressing modes are well suited to compilers and operating systems. The flexibility and simplicity of the Z80,000 provide a simple solution to hardware and software system design.

* Anil Patel is not currently at Zilog, Inc.

OVERVIEW

The Z80,000 32-bit microprocessor integrates major system functions on a single chip while also providing extensive system design flexibility. The microprocessor offers (1) Full 32-bit address and data paths; (2) on-chip instruction and data cache; (3) on-chip, demand-paged virtual memory management; (4) an execution rate of 1 to 12.5 million instructions per second (MIPS), and (5) burst transfer capability, providing enhanced bus bandwidth. A six-stage pipeline is incorporated to enhance throughput and to maximize the effectiveness of the on-board cache and memory management unit. Additional features include (1) The largest register file of any microprocessor (16 32-bit general-purpose registers), (2) nine general addressing modes, (3) a powerful instruction set directed toward support of high-level language programming, and (4) memory mapped I/O.

The Z80,000 CPU is an upward compatible extension of the Z8000 family of microprocessors and can be used with the complete line of Z8000 peripheral and software support.

ARCHITECTURE

The Z80,000 uses 32-bit logical addresses to access directly four billion bytes of memory in each of four address spaces. Separate address spaces are provided for system and normal modes and for instructions and data. Programmers have a choice of three different address representations in accessing memory space (Figure 1). Two bits in the flag and control word select compact, segmented, or linear addressing modes.

Compact mode uses a 16-bit address and is compatible with the Z8000 nonsegmented mode. Programs using less than 64k bytes of address space can take advantage of the compact mode's dense code and efficient use of base registers.

For larger programs, many applications prefer to use segmented mode, in which individual objects are allocated to separate protected segments. During address calculation the segment number is unchanged and only the offset is affected. The Z80,000 provides two segment sizes, which allow the flexibility of having both 128 segments of 16 Mbyte maximum size, and 32,728 segments of 64 Kbyte maximum size. The most significant bit of the logical address selects between the two segment sizes.

The third mode supported by the Z80,000 is 32-bit linear addressing. Applications requiring the flexibility of a large linear address space without the formal structure of segmentation can use the linear addressing mode.

A separate system mode, with its own stack pointer and protected address spaces, supports operating systems. Further, some instructions are privileged and execute only in system mode. Thus, systems software and resources are protected from normal mode programs, which can use the System Call instruction and traps to request operating system services. Operating systems also benefit from instructions aiding system call processing, context switching, and resource allocation and process synchronization.

The instruction set of the Z80,000 CPU is oriented toward compilation of high-level-language programs into compact, efficient code. The register file includes sixteen 32-bit general-purpose registers, which can hold data accumulations, index values, and memory addresses. Nine general addressing modes provide efficient access to the many types of data structures commonly used. Further, these can be combined with operations on a variety of data types (including 8-, 16-, and 32-bit integer and logical values, bits, bit fields, packed BCD bytes, and dynamic length strings) to provide a powerful instruction set. Additionally, compilers are aided by instructions for procedure linkage, array indexing, integer conversions, and other common operations.

The Z80,000 also includes extensive trap facilities for runtime error detection and software debugging, thus enhancing system development and reliability.

MEMORY MANAGEMENT

Features offered by the Z80,000's paged memory management unit include

1. Translation of logical addresses to physical addresses
2. Memory access protection
3. A translation-lookaside buffer (TLB)
4. Protected access to memory-mapped I/O devices

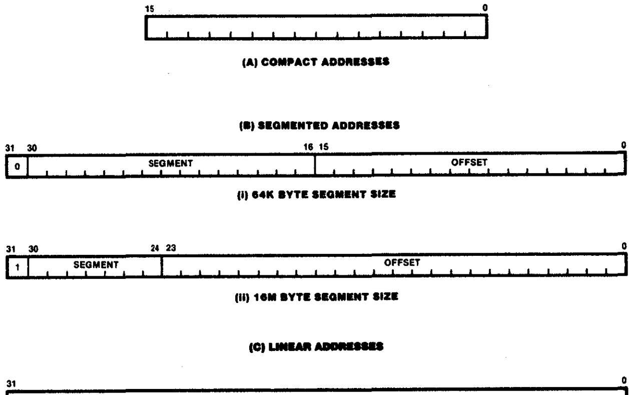


Figure 1—Address representations

Demand-paged virtual memory is easily implemented without special software fixup routines or storage of internal state following address translation faults. The implementation is accomplished through early detection of translation faults resulting in the ability to restart all instructions efficiently.

The Z80,000 implements a three-level address translation process. Once the operating system creates the translation tables in memory and initializes the on-chip translation table descriptor registers, the Z80,000 automatically references the Level 1, Level 2, and page tables to perform address translation and access protection (Figure 2).

Access protection information is encoded in a four-bit protection field at any level of the translation process. The protection field is defined for all translation table descriptors and translation table entries, but only one protection field is selected during an address translation. This allows access protection to be done at the page level, segment level, or even a mixture of these two approaches.

It is possible to reduce the number of levels of translation required by specifying in the table descriptor register that either or both Level 1 and Level 2 tables should be omitted during the translation process. Skipping the Level 1 tables is useful when only a 24-bit address space is needed; skipping the Level 2 tables can be used for Z8000-compatible programs; skipping both Level 1 and Level 2 tables is useful for compact mode programs. When a level is skipped during address translation, the corresponding bits in the logical address are ignored. In addition, the size of the tables can be reduced by specifying, in the table entries, the size of the next level tables in increments of 256 bytes; or by specifying that the entries and the tables they point to are to be considered invalid.

Delays associated with referencing translation tables are minimized by using a special on-chip buffer called the translation lookaside buffer (TLB). Logical addresses and their corresponding physical addresses are stored in the TLB as they are translated through the translation tables. Subsequent accesses to the same page do not require translation through the memory resident translation tables, but are simply retrieved from the TLB (Figure 3). A translation through the tables is performed only when an entry does not exist in the TLB (a TLB miss). The least recently used entry is then replaced with the new address and its translation. The TLB can hold the 16 most recently referenced pages.

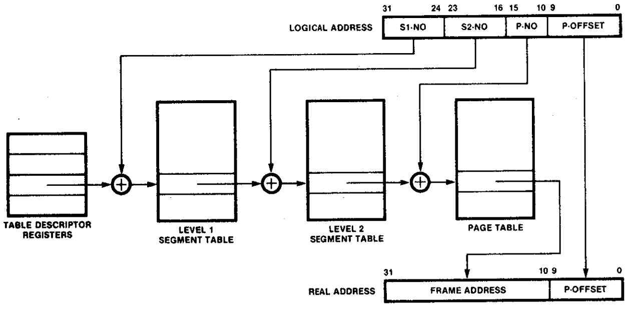


Figure 2—Address translation (The multiple fields in a logical address point to entries in several tables when the Z80,000 translates a logical address into a real address.)

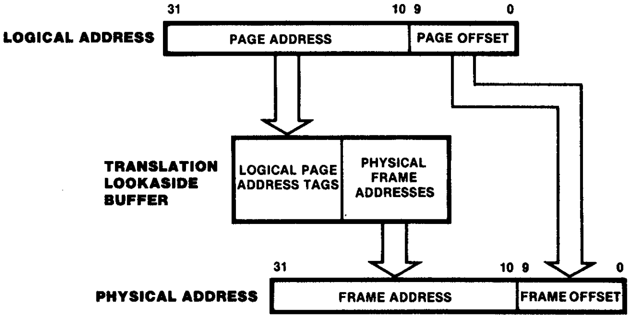


Figure 3—Address translation using the TLB

Bit 31 of the translated physical address selects either memory-mapped I/O space or memory space. If translation is enabled, any reference to I/O space carries I/O status and timing for the reference. The address translation process therefore allows normal mode (user) programs direct protected access to I/O devices without operating system overhead.

CACHE

The six-stage pipelined architecture of the Z80,000 incorporates an on-chip cache to buffer the high execution rate of the pipeline from relatively slow memory access rates. The pipeline may require two memory fetches during each processor cycle, one for the instruction fetch stage, and one for the operand fetch stage. Since main memory fetches may normally require two processor cycles, the pipeline would be idle if all references had to access main memory. The on-chip cache, though, can be accessed every clock cycle, thus satisfying most memory references without external bus transactions.

The cache effectively decouples the internal execution rate of the Z80,000 from the external bus transfer rate. Hence the system designer is able to make different memory subsystem cost-performance tradeoffs independent of the CPU. The Z80,000 bus interface can then be configured to match the CPU's clock rate to the rest of the system. This also enables easy incorporation of future higher performance versions of the Z80,000.

Copies of the most recently referenced memory locations are stored in cache. During a memory access, the address of the desired memory location is presented to the cache tag memory and if there is a match (a cache hit), then the data are read from the cache and no external memory transaction is performed. The hit ratio is typically 90% for instructions and 75% for operand fetches. If there is not a match (a cache miss), the CPU generates an external bus transaction to fetch the data from main memory, and the data are placed in the cache. The pipeline stage requesting the missing data waits until the external transaction is complete; the other stages can continue to operate.

The cache stores 16 blocks of information from main memory. Each block contains 16 bytes, providing 256 bytes of storage. Associated with each block of cache data is a 28-bit tag field. This tag field contains the 28 most significant bits of

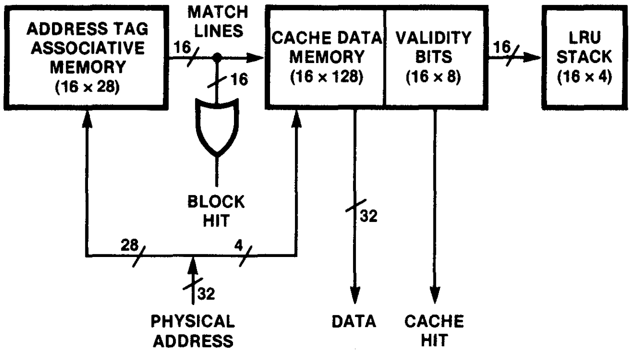


Figure 4—Cache operation

address corresponding to the block. The lower four bits of an address select the desired byte, word, or longword within the block. Each word within a block also has an associated Valid bit indicating whether the cache is holding a valid copy of the corresponding memory location (Figure 4).

When a cache miss occurs, a least recently used (LRU) algorithm is used to select a cache block to be replaced. If an instruction fetch causes a cache miss, the CPU fetches the missing instruction from memory. In addition, if the memory supports burst transactions, the balance of the block is prefetched. When a cache miss occurs on an operand fetch, then the CPU fetches only the missing data from memory. Missing operands are selectively prefetched only for instructions for which further operand fetches are anticipated.

For references which require operand stores, the data are always written to main memory. The cache is also updated if it contains the addressed location; otherwise it is unaffected. This mechanism, called writethrough, ensures that main memory always holds up-to-date data. In systems where multiple processors may share writable storage, the address translation tables allow individual pages to be marked noncacheable.

Unlike some processors, which cache only instructions, the caching mechanism of the Z80,000 can selectively cache instruction fetches and/or operand fetches. While particular applications for the Z80,000 may choose to cache only instructions, caching data along with instructions typically improves cache performance by 20%.

BUS STRUCTURE

To meet particular cost and performance objectives, the system designer can choose the memory data width, access time, and bus bandwidth simply by setting the appropriate bits in the hardware interface control register (HICR). Increased bus bandwidth can be achieved by taking advantage of the optional burst transfer capability fo the Z80,000 bus interface. The Z80,000 also provides support for different types of multiprocessor configurations.

The HICR specifies certain characteristics of the hardware configuration surrounding the CPU, including bus speed, memory data path width, and number of wait states automatically inserted. The bus speed is chosen to be either ½ or ¼ of the clock frequency. The memory data path width can be specified separately for the upper and lower portions of memory space (M_0 and M_1) as 16 or 32 bits. The number of Wait states automatically inserted by the CPU for references to the different memory and I/O spaces (M_0, M_1, I/O_0, I/O_1) may also be separately specified. Thus, a system can easily accommodate a slow, 16-bit-wide bootstrap ROM in one region and a fast, 32-bit-wide RAM in another region.

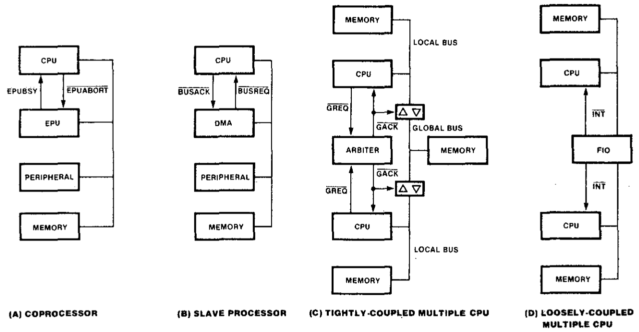
Burst memory transactions use multiple Data Strobes following a single Address Strobe to transfer data at consecutive memory addresses. The transactions are controlled by the BRST and BRSTA signals, which allow either the CPU or memory to terminate a burst transaction after any number of transfers. The CPU's protocol supports the use of interleaved memory systems and allows the system designer to easily take advantage of nibble-mode RAMs. The CPU uses Burst transactions to prefetch a cache block on an instruction fetch cache miss. They are also used to fetch or store operands when multiple transfers are necessary, as with unaligned operands, string instructions, Load Multiple instructions, and loading of program status.

The CPU provides support for interconnection in four types of multiprocessor configurations (Figure 5):

1. Coprocessor
2. Slave processor
3. Tightly-coupled multiple CPUs
4. Loosely-coupled multiple CPUs

Coprocessors work concurrently with the CPU to execute a single instruction stream, using the Extended Processing Architecture facility. This allows extension of the Z80,000 architecture to include floating point operations and other specialized functions. Additionally, the processing speeds offered by extended processing units (EPUs) optimized for particular operations, such as the Z8070 Arithmetic Processing Unit, can provide significant performance improvements. The signal EPUBSY is dedicated as a synchronization signal for connection with coprocessors. In systems supporting the functionality of an extended processing unit without the actual EPU present, the EPU instructions are trapped and emulated in software.

Slave processors, such as the Z8016 Data Transfer Controller, perform DMA functions asynchronously to the CPU. The CPU and slave processor share a local bus, of which the CPU



(A) COPROCESSOR (B) SLAVE PROCESSOR (C) TIGHTLY-COUPLED MULTIPLE CPU (D) LOOSELY-COUPLED MULTIPLE CPU

Figure 5—Multiprocessor configurations

is the default master, using the CPU's $\overline{\text{BUSREQ}}$ and $\overline{\text{BUSACK}}$ lines.

Tightly coupled multiple CPUs execute independent instruction streams and generally communicate through shared memory located on a common (global) bus using the CPU's $\overline{\text{GREQ}}$ and $\overline{\text{GACK}}$ lines. Each CPU is default master only of its local bus, while an external arbiter chooses the global bus master. The CPU also provides status information about interlocked memory references so that bus control is not relinquished during an indivisible operation such as Test and Set or Increment Interlocked.

The Z80,000's I/O and interrupt facilities support loosely coupled multiple CPUs, which generally communicate through a multiported peripheral, such as the Z8038 Z-FIFO I/O Controller.

PIPELINED ARCHITECTURE

The Z80,000 has a six-stage pipelined architecture. The following functions are performed at each of the stages:

1. Instruction fetch
2. Instruction decode
3. Address calculation
4. Operand fetch
5. Instruction execution
6. Operand store

For simpler instructions all stages of the pipeline (except for operand store) are completed in one processor cycle (two clock cycles) (Figure 6). Therefore at 25 MHz the peak instruction execution rate is 12.5 MIPS. In practice the actual instruction execution rate is about one-third the peak rate, depending on the instruction mix and system configuration.

The on-chip cache is time-multiplexed between instructions and data, allowing for the occurrence of instruction and data accesses within the same processor cycle. The pipeline then flows smoothly without having two stages competing for the same resource. Similarly, there are two ALUs, one for address calculation and one for instruction execution. This eliminates sharing a single ALU between the address calculation stage and the execution stage. The on-chip TLB also helps smooth the pipeline flow, as the address translation is done in the address calculation stage without having to go off-chip. The operand storage stage is the only stage requiring several cycles. Performance is not seriously degraded, however, because the store operation can usually overlap with the processing of other instructions.

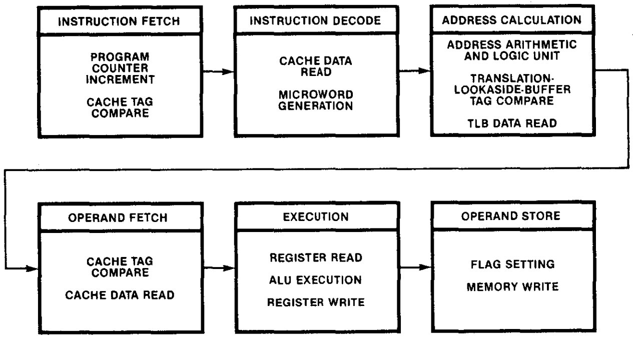


Figure 6—Pipes (Six stages of pipelining boost performance in the 32-bit Z80,000 central processing unit. Since accesses to external memory would not be fast enough to keep the pipe full, the Z80,000 uses on-chip cache store to buffer the pipeline.)

Z80,000 CPU PERFORMANCE

Cache memory and the pipelined structure cause the performance evaluation of the Z80,000 CPU to be complex. The best approach is separation of instruction processing time into a sum of three components: execution time, pipeline delays, and memory delays. Performance was evaluated by statistically measuring activities of ten C language programs, and then performing a computer simulation of the cache, translation-lookaside buffer, and pipeline mechanisms.

The execution time for an instruction is the number of cycles required to execute the instruction if no other delays, such as a cache miss or register interlock, occur. Common instructions, such as loading a register with a word operand specified by a base-register-plus-displacement addressing mode, execute in one processor cycle (two clock cycles), but the average instruction execution time is 1.3 processor cycles.

Pipeline delays are caused by branch instructions, register interlocks, and other miscellaneous delays. The most significant of these delays is the delay due to branch instructions. When a branch is taken, instructions in the pipe behind the branch instruction are flushed. Unconditional branches introduce a delay of two processor cycles. Conditional branches cause a three-processor-cycles delay if the condition is met, and no delay if the condition is not met. The average delay due to branches is 0.5 processor cycle per instruction.

Another significant pipeline delay is register interlock. Whenever the execution stage modifies a register to be used in a subsequent instruction as an address register, the address calculation must be held up (interlocked) until the execution is complete. The interlock ensures that the proper register value is used in the address calculation. A register interlock occurs for 11% of instructions, causing a 0.19 processor cycle delay, and a cache reference interlock occurs for 6% of instructions, causing a 0.11 processor cycle delay. Therefore the total average pipeline delay is 0.9 processor cycle per instruction.

Memory delays are caused by cache misses and TLB misses. When the processor fetches an instruction or operand for which a corresponding entry in the cache or TLB does not exist, a reference to main memory is generated. The average delay due to these memory transfers is calculated assuming a 32-bit data path, a cycle time of two processor cycles for read transactions, a cycle time of three processor cycles for write transactions, and support of burst transfers. An average delay per instruction due to cache misses is 0.4 processor cycle, and an average delay per instruction due to TLB misses is 0.5 processor cycle. Therefore, the total memory delay with a Memory Management Unit is 0.9 processor cycle. The memory delay without memory management is 0.4 processor cycle.

Instruction processing time Ti = Execution delay + Pipeline delay + Memory delay. Therefore, Ti = 1.3 + 0.8 + 0.9 = 3.0 processor cycles. Ti without MMO = 1.3 + 0.8 + 0.4 = 2.5 processor cycles. At 25 MHz, this corresponds to 4.2 MIPS. At 25 MHz, without MMO, the CPU's average sustained performance is 5 MIPS.

The above analysis assumes that these delays occur in sequence. In reality simultaneous delays cancel each other, and consequently the actual CPU performance may be better than the calculated CPU performance.

Introducing the MC68020: Performance and flexibility improvement through redesigned hardware and software enhancements*

by CLARA I. SERRANO
Motorola, Inc.
Austin, Texas

ABSTRACT

The MC68020 microprocessor is the full 32-bit implementation of the M68000-family architecture providing 4 gigabytes of linear address space. The MC68020 is upward user object-code compatible with the MC68000, MC68008, MC68010, and MC68012. As far as the user can determine, the architecture of the MC68020 is merely an extension of earlier processors in the family. However, the MC68020 has many new features invisible to the user program that enhance speed of operation, and many new software features that provide added flexibility for the system designer and programmer. Some of the hardware enhancements include additional supervisor registers, increased parallelism, an on-chip cache, a deeper instruction pipe, 3-bus-cycle reads and writes, a barrel shifter, three 32-bit ALUs, and an asynchronous dynamic bus-sizing interface. Software enhancements include a richer instruction set with added support for high-level languages, additional internal registers, a very general coprocessor interface, additional addressing modes, more sophisticated interrupt control, added exception-processing capabilities, and added privilege levels for memory management support; this is in addition to all the virtual memory/virtual machine support available with the MC68010 and MC68012, and the reliability provided by the exception-handling structure of all the other processors in the family.

Two new M68000 coprocessors, the MC68881 Floating Point Coprocessor and the MC68851 Paged-Memory Management Unit, can be directly interfaced with the MC68020. The MC68881 carries out all internal calculations to 80 bits of precision, and conforms to the IEEE Floating Point Specification. The MC68851 provides very flexible demand-paged memory management support and performs logical-to-physical address translations in 45 nanoseconds.

*Preliminary version. © 1984 by Professional Educational International, Inc., in the *Proceedings of the National Communications Forum*. Printed by permission.

INTRODUCTION

The M68000 family of microprocessors and peripherals has provided extensive processing power and flexibility to system designers working on applications ranging from low-cost microcomputer consumer products to sophisticated multi-user engineering workstations. As technology has advanced and system integration progressed, a new high-performance microprocessor market has evolved—that of the 32-bit world. Users and designers in 32-bit environments typically wish to emulate the features of mini/mainframe computers, but at microprocessor sizes and prices. Motorola's 32-bit implementation of the M68000 architecture, the MC68020, is an attractive solution for 32-bit applications that require raw processing power plus comprehensive hardware and software interfaces.

Two very important design concepts dominated the specification of the M68000 family architecture, and now the implementation of the MC68020; the first is performance or simply execution speed; the second is generality and extensibility, providing the system designer with a general purpose architecture with which to build a sophisticated system that solves the problem at hand with minimal effort and cost. These concepts are obviously related in that the most flexible of architectures allows the designer to implement objectives efficiently.

The new features of the MC68020 are discussed and highlighted from a systems standpoint.

M68000 FAMILY OVERVIEW

The M68000 is considered to be a 16/32-bit architecture, because all microprocessors in the family have 32-bit internal registers, can perform 32-bit operations, but do not all bring the 32-bit address and data paths directly to the outside world. This family of processors consists of the following:

1. The MC68000 with 24 address lines and 16 data lines
2. The MC68008, a reduced-cost part with 20 address lines and 8 data lines
3. The MC68010 Virtual Memory/Machine processor with the same addressing range of the MC68000, but with the added registers and hardware to allow full virtual-memory support
4. The MC68012, an extension of the MC68010 with 7 additional address lines, and now
5. The MC68020, the full 32-bit implementation of the family architecture

An important architectural feature of the entire family of processors is that the address space for each CPU is linear (not segmented).

Asynchronous Bus Structure

The M68000 family is a very popular choice for system designers interfacing a CPU with various speed memory and peripheral devices because of the performance and flexibility available with an asynchronous, non-multiplexed interface to these devices. With M68000 microprocessors, the access timing of the processor is dynamically controlled on each bus cycle by the device being accessed via a handshake signal called *DTACK* (Data Transfer Acknowledge). Thus with the fastest memory devices, a *no wait-state* system can be designed such that memory reads and writes only take four clock cycles. Another portion of the bus structure dynamically gives up the external bus to other requesting bus masters (i.e., DMA Controllers, other CPUs etc.) via the bus arbitration pins BR (Bus Request), BG (Bus Grant), and BGACK (Bus Grant Acknowledge), thereby allowing maximum external bus utilization.

Exception Processing

The asynchronous bus structure also handles hardware failures and improper memory accesses in a very straightforward manner. If data is not read or written correctly to memory (for example, if error detection and correction logic finds an error, or if a page fault occurs in a virtual-memory system), the external circuitry asserts BERR (Bus Error), and the microprocessor enters exception processing to handle the error gracefully. With the MC68010, MC68012, and MC68020, the simultaneous assertion of BERR and HALT causes the processor to rerun the current cycle, providing very quick recovery from a memory error or other predefined conditions.

Interrupts

External interrupts cause control to be passed directly to an exception-handler routine. The efficient handling of interrupts makes the M68000 family the logical choice for applications requiring real-time processing. Other exceptions can be initiated by software (via TRAP instructions, for example), and by hardware (tracing, etc.) yielding more than 200 (230 for the MC68020) unique exception routines that can automatically be called by the CPU when the corresponding conditions exist.

Instruction emulation

Another kind of exception occurs when the opcode of an instruction is not recognized by the processor. In this case, the

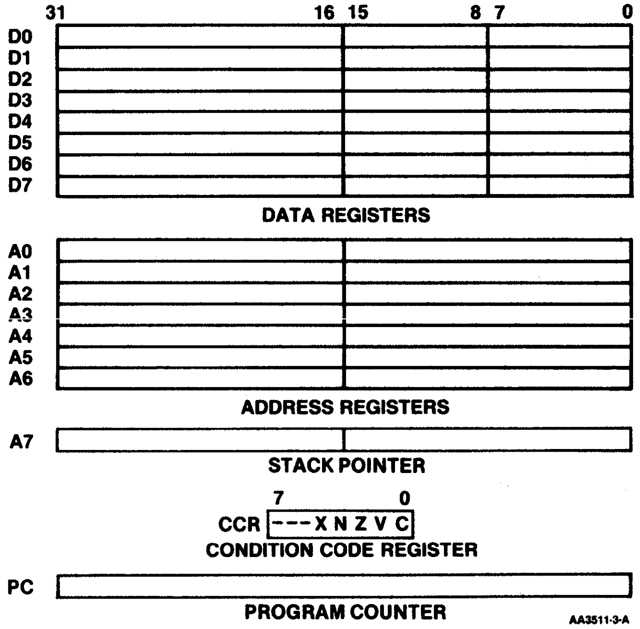


Figure 1—User programming model for all M68000 microprocessors

CPU automatically passes control to an illegal instruction-handler routine, providing the capability to simulate user-defined instructions in software.

Register Set

The basic user programming model for all microprocessors in the M68000 family is shown in Figure 1. This model includes 16 general-purpose address and data registers, all 32 bits wide; a condition code register with carry, overflow, zero, negative, and extend bits; and a program counter providing direct program access to the full addressing range of each of the processors. The data registers may contain byte, word, or longword operands; the address registers may contain word or longword pointers with A7 as the default user stack pointer.

HARDWARE ENHANCEMENTS

MC68000 Supervisor Registers

Each of the M68000 processors has a second set of registers that can only be accessed in the *supervisor* state. This privilege distinction is particularly useful in an operating system environment where the user should not have direct access to operating system handling information. In the case of the MC68020, this set of registers includes two supervisor stack pointers, the master stack pointer and the interrupt stack pointer (see Figure 2). These two stack pointers facilitate multi-tasking control by allowing each task to have its own master stack containing control information relevant to that task. For example, when an interrupt occurs, thereby signalling the need for a task switch, the interrupt handler simply loads the master stack pointer, which points to the control block of the new task, without having to transfer all of the interrupt-related information from the previous task's context, as this information is kept on the interrupt stack (see Figure 3).

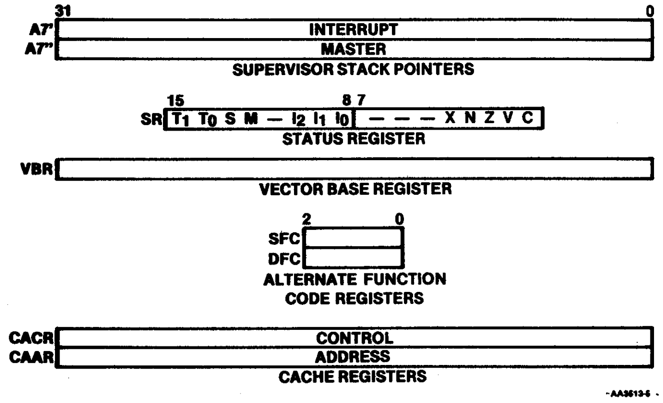


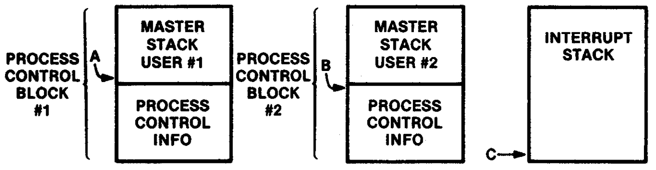
Figure 2—MC68020 supervisor programming model

The status register for the MC68020 contains the condition code bits (present in the user model), and the interrupt mask encoded in I2, I1, and I0. The *M* bit determines if the master or interrupt stack is being used by the supervisor, and the *S* bit determines if a program is executing in the user or supervisor state. The MC68020 also has two *trace bits* (T1 and T0) that allow a software debugger to trace on every instruction boundary, or on changes in program flow (e.g., a BRANCH instruction).

The Vector Base Register allows the exception vector table (containing the pointers to exception handler routines) to be arbitrarily relocatable, and therefore supports multiple exception vector tables. The alternate function code registers (SFC and DFC) allow the supervisor to access any address space by explicit manipulation of the function codes. Finally, the cache control registers (CACR and CAAR) allow the supervisor to manipulate the on-chip instruction cache in software.

Enhanced Bus Structure

Although the architecture of the MC68020 appears to be an extension of the MC68010 and MC68012, it is actually a com-



WITH M BIT SET:

— ALL TASK RELATED EXCEPTIONS (NON-INTERRUPTING EXCEPTIONS) ARE PLACED IN USER'S PCB

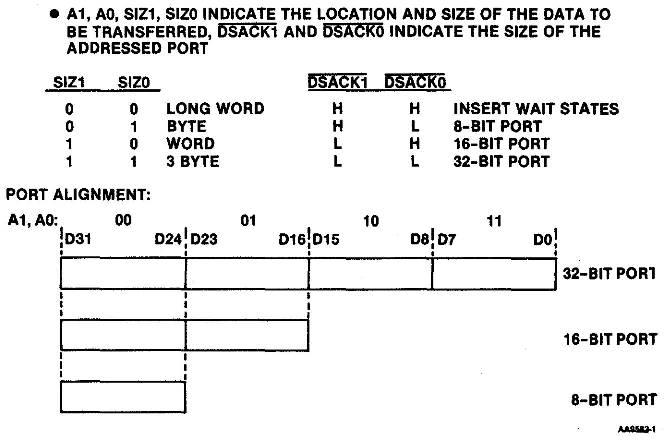
— INTERRUPT FRAME IS PLACED IN BOTH USER'S PCB AND INTERRUPT STACK (ON FIRST INTERRUPT ONLY)

— ALL EXCEPTIONS WHILE PROCESSING INTERRUPTS ARE PLACED ON INTERRUPT STACK

— TASK CHANGES BY RELOADING MSP (POINTER A OR B) AND SETTING M BIT

AA9676

Figure 3—Master and interrupt stacks



- A1, A0, SIZ1, SIZ0 INDICATE THE LOCATION AND SIZE OF THE DATA TO BE TRANSFERRED, $\overline{\text{DSACK1}}$ AND $\overline{\text{DSACK0}}$ INDICATE THE SIZE OF THE ADDRESSED PORT

SIZ1	SIZ0		$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	
0	0	LONG WORD	H	H	INSERT WAIT STATES
0	1	BYTE	H	L	8-BIT PORT
1	0	WORD	L	H	16-BIT PORT
1	1	3 BYTE	L	L	32-BIT PORT

Figure 4—MC68020 data transfers

plete redesign, internally; it has a redesigned bus structure that can perform read and write cycles in as little as three clock cycles for a *no wait-state* system; it can also move 32 bits of data in one bus cycle. Furthermore, full hardware compatibility has been maintained with the reduced data buses of earlier processors in the family via the new dynamic bus-sizing capability of the MC68020. The processor now dynamically decides how much data to transfer to or from a peripheral, depending on the handshaking it receives from the peripheral. DTACK has been replaced with DSACK1 and DSACK0 (Data Transfer and Size Acknowledge), two inputs that control access timing for a given transfer and tell the CPU the size of the port being accessed by the encoding on these two lines. For example, the MC68020 automatically performs the operations required to transfer a longword (32 bits) to an 8-bit port without prior knowledge of the port size (see Figure 4). In addition, the MC68020 has no restrictions concerning alignment of operands in memory; word and longword operands may be aligned on any byte boundary. For consistency with earlier processors and to optimize performance, instruction words are still 16 bits long and must reside on word boundaries.

Parallelism

Parallelism within the chip has been expanded in various ways. The instruction prefetch mechanism implements a 3-stage pipeline instead of the 2-stage pipe in earlier family members. Thus as one instruction is being completed (for example, the final *read from* or *write to* external memory), the next instruction begins execution internally. The subsequent instruction may be in the decode stage and a fourth word could be in the process of being fetched from the cache.

In order to make parallel execution most efficient, 32-bit wide multiple data paths were implemented internally. The many possible combinations of overlap have made instruction timing parameters very difficult to calculate and very application dependent. Under certain conditions, this increased degree of overlap may actually cause an instruction to effectively execute in *zero time.*

Instruction Cache

The MC68020 was designed to include a very high-speed cache memory on-chip for storing the instruction stream accesses in case they are accessed again. Mainframe computers have historically implemented small high-speed memories in order to store instructions and data that are accessed frequently due to the phenomenon known as locality of reference (i.e., the looping nature of most assembly language programs). The MC68020 is the first 32-bit microprocessor to incorporate the advantages of on-chip cache memory.

As shown in Figure 5, the cache is direct-mapped (i.e., each address maps into a unique location in the cache) with A7–A2 serving as an index into the tag store. The tag consists of the upper 24 address bits, Function Code 2 (to protect supervisor programs from user accesses), and a valid bit (ensuring that a valid instruction has been loaded into that location of the cache). Thus if an instruction prefetch is initiated, and the address in the tag (selected by the index) matches with the prefetch address, a *hit* is said to have occurred and the instruction is then fetched from the cache. When a cache *miss* occurs, and the cache is enabled, the CPU continues its external memory access, and loads the new instruction into the cache for possible future reference. Data is not stored in the cache, so that other bus masters may manipulate data areas without the possibility of the CPU then having "stale data" in its cache.

The supervisor controls the cache via the two cache control registers. It can enable, clear, or freeze the cache, or clear a particular cache entry. In the case of a clear entry operation, the cache address register (CAAR) is used to specify the entry to be invalidated. An external Cache Disable pin (CDIS) operates independently of the cache enable bit in the cache control register. CDIS can disable the cache to aid in software debugging.

The MC68020 cache enhances overall system performance as follows:

1. When an instruction is found in the cache, it is accessed in only two clocks (a 33% improvement over a no wait-state external prefetch).
2. When programs are executing out of the cache, the available external bus bandwidth is increased, thereby allowing other bus masters more access to the buses.
3. Because instruction words are specified as 16-bit quantities, and instruction prefetches are always made as longword reads, almost every other instruction word is sure to be found in the cache.
4. Because of the parallelism designed into the MC68020, the internal cache bus is isolated from the external bus so that instructions can be fetched from the cache at the same time that external data accesses are made.

Internal Arithmetic and Control Features

The MC68020 has three 32-bit Arithmetic and Logic Units (ALUs), which may be employed independently or in unison, again adding to the performance factor. A barrel shifter was also included to allow high-speed shift and rotate operations

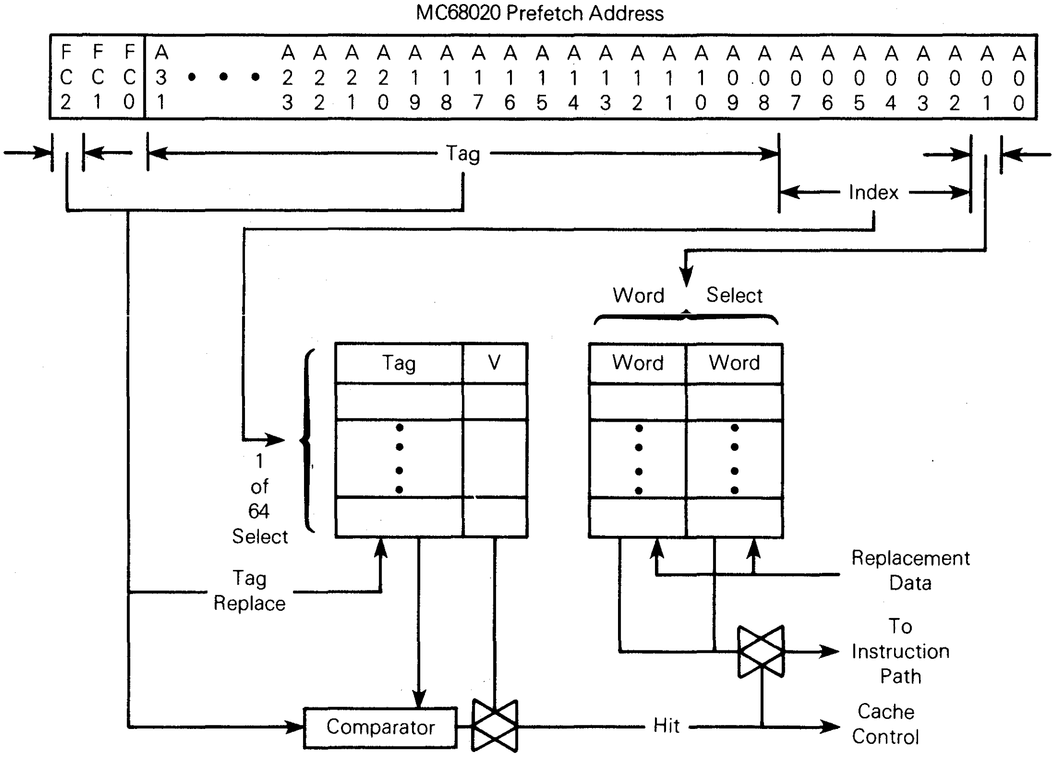


Figure 5—MC68020 on-chip cache organization

to execute in a fixed amount of time (independent of the number of shifts), and a PLA was implemented in the decode logic to insure expeditious recognition of coprocessor instructions.

SOFTWARE ENHANCEMENTS

The instruction set of the MC68020 is a superset of the instructions available with the earlier M68000 processors. The software enhancements in the MC68020 emphasize further flexibility and increased support for sophisticated operating systems and structured high-level languages.

Addressing Modes

The addressing modes have been expanded to allow 32-bit base addresses, indexes, base displacements and outer displacements to be specified in a single mode, and sizing and scaling of the index register by two, four, or eight. Sophisticated memory-indirect addressing modes allow for efficient generation of addresses to satisfy most complex programming structures. Intermediate memory accesses can be made in the calculation of an effective address, and the program counter may be used as a base displacement in any addressing mode. Post-increment and pre-decrement modes are present (as with other M68000 systems), thus allowing the user to create stacks and queues anywhere in memory.

Trap Exceptions

Supervisor exception processing can be initiated in software by the use of TRAP # or TRAPcc instructions. The programmer may pass control to a supervisor handler routine with 16 TRAP # exceptions, or with 16 trap-on-condition possibilities. The format of the TRAP instructions now allows two to four bytes of additional user-specified information to be passed to the trap handler as well.

Breakpoints

A BKPT instruction causes a Breakpoint Acknowledge cycle to be initiated. This bus cycle places a user-specified value of 0–7 on A4–A2. If the cycle is terminated by DSACKs, the value on the data bus is used to replace the breakpoint opcode in the instruction stream; if terminated by BERR, an exception is taken. The breakpoint instructions allow for sophisticated debug environments and real-time tracing of cache-resident routines.

Bit Field Support

A new data type called a *bit field* has been defined to be used with new bit field manipulation instructions. A bit field is a string of one to 32 bits that can reside in a data register or anywhere in memory with no restrictions on boundaries (i.e.,

<ea> {OFFSET: WIDTH}

<ea>—SPECIFIES THE BASE LOCATION OF THE FIELD

OFFSET—NUMBER OF BITS FROM <ea> TO THE START OF THE FIELD. MAY BE AN IMMEDIATE VALUE OF 0-31 OR IN A DATA REGISTER WITH A VALUE OF -2^{31} TO $2^{31}-1$

WIDTH—NUMBER OF BITS IN THE FIELD, MAY BE AN IMMEDIATE VALUE OR IN A DATA REGISTER IN THE RANGE OF 1 TO 32

BIT FIELDS MAY TRAVERSE ANY BYTE BOUNDARIES

EXAMPLE:

BFEXTU (REC5, A3, D3.W*4){D4:24}, D0

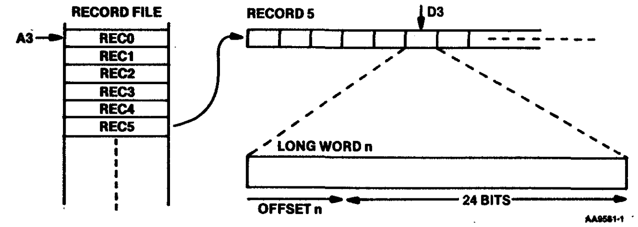


Figure 6—Bit field specification

a 32-bit bit field may cross up to 4-byte boundaries). It is specified by a base location, an offset from that base, and a width as shown in Figure 6. The range of instructions extends from bit field insert (BFINS), signed and unsigned bit field extract (BFEXTS, BFEXTU), and bit field find first one (BFFFO), to bit field test, change, set and clear (BFTST, BFCHG, BFSET, and BFCLR).

Binary Coded Decimal (BCD) Operations

Two new instructions, PACK and UNPACK, allow decimal data to be stored in a condensed form providing expansion when necessary. In effect, the PACK instruction condenses two BCD digits into one byte, and the UNPACK instruction expands the operand so that it corresponds to one digit per byte. When using PACK or UNPACK, the programmer may specify a constant to be added to support EBCDIC, ASCII, or any other number representations.

Module Loading

The CALLM instruction references a module descriptor with control information for that module, verifies legal entry into the module, and stores the module state in the module stack frame. If the module is not available, an exception occurs, thereby returning control to the supervisor and allowing it to dynamically load the module. The RTM instruction recovers the previous module state from the module stack frame and returns control to the calling module.

CAS, CAS2, CHK2, and CMP2

The CAS and CAS2 instructions are indivisible read-modify-write *compare and swap* operations included for more flexible multi-processor interfacing. The CMP2 and CHK2 instructions provide more flexible bounds-checking, allow setting condition codes on a compare, or cause an exception if a check fails.

COPROCESSOR INTERFACE

Extending the software capabilities of the MC68020 is achieved through its powerful coprocessor interface. A general-purpose protocol was established for CPU/coprocessor communication so that designers may customize systems with any type of coprocessor.

Coprocessor instructions are characterized by a leading "1111" in the opcode, and as mentioned earlier, an internal PLA detects a coprocessor instruction very early in its decode cycle. The CPU communicates with the coprocessor by placing access information on the address bus, including a coprocessor ID on A11–A9, selecting one of eight possible coprocessors. The coprocessor responds to the CPU by means of a coprocessor *register* indicating that it is busy, or that it is ready for more information. The possibilities of coprocessor responses are shown in Figure 7.

Coprocessors are classified as bus masters or bus slaves. A bus master type of coprocessor (for example, a paged memory management unit) can take control of the external buses; a slave-type has no bus control hardware.

Floating Point Support

The MC68881 Floating Point Coprocessor interfaces to the MC68020 as a coprocessor, or it can interface to other CPUs in the M68000 family as a peripheral. It fully implements the IEEE Floating Point Specification (Draft 10.0), and carries out all internal calculations to 80 bits of precision.

- PROCESSOR SYNCHRONIZATION
  - — BUSY FROM PREVIOUS INSTRUCTION
  - — BUSY WITH CURRENT INSTRUCTION
  - — PROCEED WITH NEXT INSTRUCTION, IF NO TRACE
  - — PROCEED WITH NEXT INSTRUCTION, IF TRACE ENABLED
  - — PROCEED WITH EXECUTION, CONDITION TRUE/FALSE

- INSTRUCTION MANIPULATION
  - — PASS OPCODE
  - — PASS WORDS FROM INSTRUCTION STREAM

- EXCEPTION HANDLING
  - — TAKE PRIVILEGE VIOLATION IF S-BIT NOT SET
  - — TAKE PRE-INSTRUCTION EXCEPTION
  - — TAKE MID-INSTRUCTION EXCEPTION
  - — TAKE POST-INSTRUCTION EXCEPTION

- GENERAL OPERAND TRANSFER
  - — EVALUATE AND PASS <EA>
  - — EVALUATE <EA> AND TRANSFER DATA
  - — WRITE TO PREVIOUSLY EVALUATED <EA>
  - — TAKE ADDRESS AND TRANSFER DATA
  - — TRANSFER TO/FROM TOP OF STACK

- REGISTER TRANSFER
  - — TRANSFER CPU REGISTER
  - — TRANSFER CPU CONTROL REGISTER
  - — TRANSFER CPU SR AND/OR PC
  - — TRANSFER MULTIPLE CPU REGISTERS
  - — TRANSFER MULTIPLE COPROCESSOR REGISTERS

Figure 7—Coprocessor response primitives

Memory Management

Due to a design philosophy that emphasizes flexibility, the MC68020 does not confine the system designer to a particular memory management scheme. However, the MC68851 Paged Memory Management Unit is available for those systems requiring support for paged virtual memory. The MC68851 has an on-chip content addressable memory that stores multiple page descriptors (64), once again supporting very efficient task switching and real-time interrupt processing in systems using memory management. It also implements up to eight levels of program protection, and contains table search hardware to automatically replace cache entries on a translation miss.

SUMMARY

The performance of the MC68020 exceeds that of any other 32-bit microprocessor on the market for several reasons. The hardware design incorporates state-of-the-art features such as an on-chip cache, multiple ALUs, and execution overlap. However, software enhancements also contribute to performance by allowing flexible interfacing with the rest of the system and significant operating system support. Furthermore, the general-purpose coprocessor interface on the MC68020 provides for limitless expansion by allowing the implementation of special purpose coprocessors that communicate with the MC68020.

Hardware configurations and I/O protocol of the WE32100 microprocessor chip set

by MICHAEL L. FUCCIO, LAKSHMI GOYAL, and BENJAMIN NG
AT&T Bell Laboratories
Holmdel, New Jersey

ABSTRACT

In this paper, the hardware design features of the WE32100 Microprocessor chip-set are briefly summarized. That is, the hardware protocols and chip-set configurations are discussed. The WE32100 chip family provides the VLSI core of general purpose computer systems providing virtual memory and high speed floating point arithmetic. The system design features of the chip family support the software architecture of the chip-set and allow easy system interface and integration. The WE32100 chip family includes the WE32100 Microprocessor (CPU), WE32101 Memory Management Unit (MMU),[1] WE32106 Math Accelerator Unit (MAU),[2,3] WE32103 Dynamic RAM Controller (DRC), WE32104 Direct Memory Access Controller, and the WE32105 System Interface Unit (SIU).

INTRODUCTION

The WE32100 chip-set performs a multitude of general purpose computer functions. The chip-set consists of 6 VLSI chips implemented in 1.75μm CMOS technology. Typical operating frequency of all chips is 14 MHz. All chips have a TTL compatible interface and are specified with a 130 pf loading on outputs except for the WE32103 Dynamic RAM Controller (DRC), which controls large banks of DRAMS, and hence has some outputs specified for greater than 130 pf load. Each chip dissipates less than 1.5W of power.

FEATURE LIST

The WE32100 chip-set provides super mini-computer features and performance in just a few VLSI chips. Some features are described below.

WE32100 CPU Features

1. WE32100 CPU is an upward compatible version of the WE32000 (formerly BELLMAC-32A) microprocessor. The WE32100 CPU has a rich instruction set with operating system support instructions included.[4]
2. A general purpose support processor interface consisting of 10 instructions and associated I/O protocol is provided to allow for support-processor elements such as WE32106 Math Accelerator Unit.
3. The WE32100 CPU has a rich interrupt structure including a "Quick-Interrupt," "process switch interrupt," and "automatic-vector interrupts."
4. The WE32100 CPU has an on-chip instruction cache for enhanced performance.
5. The WE32100 CPU has a dynamically selectable two-word block-fetch capability for filling the instruction cache. This feature allows fetching of a 2-word block while issuing only one address. This feature reduces the MMU translation overhead for instruction fetches.

WE32101 MMU Features

1. The WE32101 Memory Management Unit has the same software model as the WE32001 memory management unit.[5,6]
2. The WE32101 MMU can operate in concert with other WE32101 MMUs to provide a virtual-address environment for the WE32100 chip-set, or can operate alone to perform the same task.
3. The WE32101 MMU supports segmented and/or paged virtual memory systems.[2]
4. The WE32101 MMU provides a PAGED/CONTIGUOUS translation indication to support the "early RAS" feature in the WE32103 Dynamic RAM Controller. This "early RAS" feature eliminates a cycle in typical DRAM accesses.

WE32106 MAU Features

1. The WE32106 Math Accelerator Unit supports fully the IEEE draft 10 floating point standard[7] and can operate via the support processor interface or as a common peripheral chip, the former providing significant performance benefits.

WE32105 SIU Features

1. The WE32105 System Interface Controller provides a flexible system interface and is a general purpose bus interface chip. This chip is useful but not essential to the operation of the chip-set.

WE32103 DRC Features

1. The WE32103 Dynamic RAM Controller supports normal READ/WRITE operations to dynamic memory including AT&T Technologies' 256K dynamic RAMs.
2. The WE32103 DRC has six programmable refresh modes and is also programmable for different access time DRAM chips.
3. The WE32103 supports page or nibble mode on DRAMS and supports "early RAS" feature for paged virtual to physical address translation.
4. The WE32103-DRC supports double and quad word read/write operations.
5. Also supported on the DRC are handshake signals for interfacing to an error detection and correction chip.
6. Dual ported memory configurations are also supported by the DRC.

WE32104 DMAC Features

1. The WE32104 DMAC provides a separate 8 bit peripheral bus in addition to a fully demultiplexed 32-bit system bus.
2. The WE32104 DMAC supports up to 4 independent channels with programmable priority levels and internal data buffering for each channel.

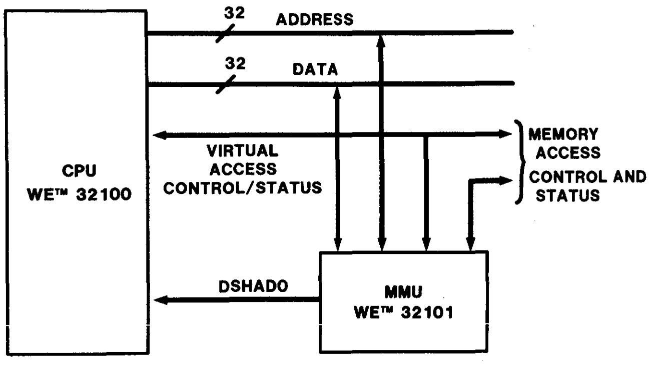


Figure 1—CPU with MMU

3. The DMAC provides the ability for command chaining.
4. The DMAC allows accesses to the peripheral bus from the system bus.

CHIP-SET CONFIGURABILITY

The WE32100 chip-set was architected for multiple configurations. The WE32100 CPU can be a stand-alone microprocessor or it can operate in a number of chip-set configurations depending upon system needs. Each chip set configuration requires no SSI "glue" to integrate the members of the chip family into a fully functional core of high performance computer systems. This "no-glue" solution provides very high functional density and saves precious board space for single-board-computer designers. Some configurations are shown in stylized form in the following figures.

Figure 1 shows a CPU with MMU. The salient features of the CPU-MMU configuration are shared address and data buses. The MMU becomes a bus master, in its own right, when it performs miss-processing to fill its internal translation caches. The MMU translation overhead is one cycle, hence a native three cycle access becomes a four cycle access with virtual to physical address translation. The MMU uses the DSHADO signal to preempt an on-going CPU access when miss processing is performed.

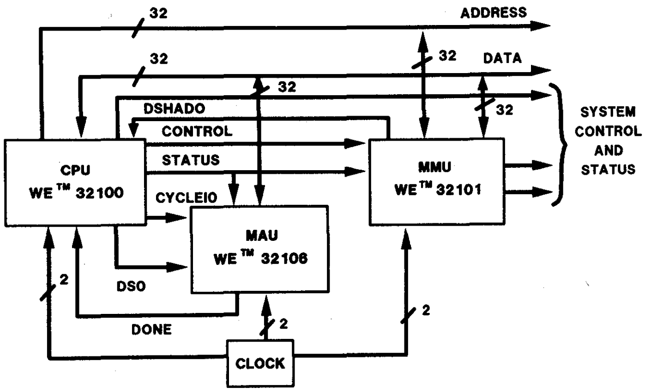


Figure 2—CPU with MAU and MMU

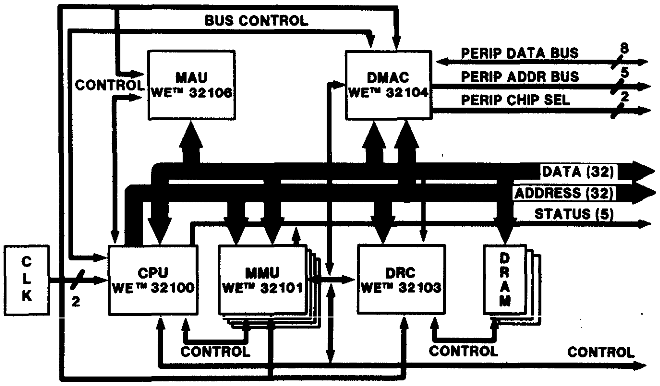


Figure 3—CPU with MMU, MAU, DMAC, and DRC

Figure 2 shows a CPU with MMU and MAU. The notable feature of the CPU-MMU-MAU configuration is, once again, the shared, non-multiplexed address and data busses. The math accelerator unit, when configured as a support processor, takes data directly off of the data bus. Floating point operand addresses are generated by the CPU.

Figure 3 shows a CPU with MMU, MAU, DMAC and DRC. This configuration shows the shared address and data busses. This configuration also shows the separate peripheral bus on the DMA controller. This peripheral bus is ideally suited to character oriented I/O such as UARTs (Universal Asynchronous Receiver Transmitter) or a local area network interface. Figure 3 also shows that multiple MMUs can provide a larger translation buffer if necessary. Furthermore, the DRC chip is shown controlling multiple banks of dynamic RAM.

Figure 4 shows a CPU with System Interface Unit. The System Interface Unit (SIU) provides many common functions which are usually implemented with SSI logic such as byte-write strobes and WAIT-STATE generation circuitry. The WE32100 chip-set can be configured with or without the SIU chip. The SIU chip can be part of all the WE32100 chip-set configurations or none of them.

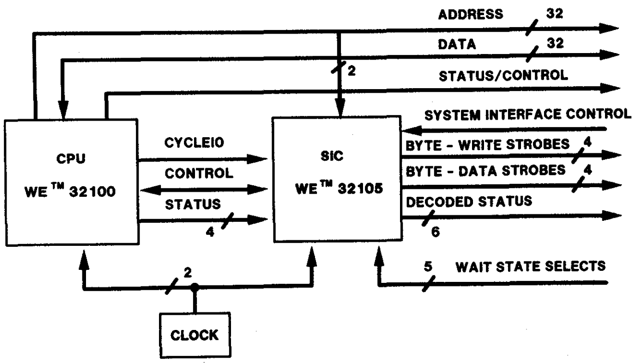


Figure 4—CPU with SIC

INTERCHIP PROTOCOL

The WE32100 chip-set has a 3-cycle (zero wait-cycle) memory access transaction with the added overhead of one cycle for the MMU to perform address translation. The block fetch feature requires five cycles (zero wait-cycle) for the two-word transfer with only one additional cycle for the MMU to perform address translation. The block fetch feature reduces the MMU translation overhead on instruction fetches. This reduction in overhead is large when there are many wait-states required to access memory. The operands for the support processor require three cycles (zero wait-cycle) to complete the support processor transaction, hence no more time is spent loading operands to the MAU than are spent by the CPU to fetch the operands. There is no overhead in loading support processor operands.

The hardware interface signals presented to the rest of the system (e.g., memory) are not unlike common microprocessors. Though the interchip protocol is essentially a synchronous protocol, the system interface is asynchronous in the sense that a 2-direction handshake between the chip-set and the external system is implemented. All asynchronous signals are sampled and doubly latched to avoid metastability of input signals. The four-cycle memory access with MMU translation is described below:

Read
Transaction
(See Figure 5)
Cycle Activity

1 At the beginning of the first cycle, the address is issued. Status indicating the size of the datum to be read is issued. Also a read indication is generated.

In mid-cycle, the virtual address strobe and a data strobe are asserted indicating that a valid address is on the address bus and that data may be put on the data bus.

2 If the address is a virtual address, it is tri-stated in this cycle to allow the translated address to be put on the bus. If an asynchronous data transfer acknowledge (DTACK) signal is present at mid-cycle then the transaction will terminate in the following cycle. If the synchronous data transfer acknowledge (SRDY0) signal is present at the end of this cycle then the transaction will terminate in the next cycle. If no data transfer acknowledge is present then the next cycle will be a WAIT-CYCLE. Note that the MMU requires one WAIT-CYCLE to perform virtual to physical address translation.

3 Without an MMU, the CPU can sample data in this cycle and terminate the transaction. With an MMU present this cycle is a WAIT-CYCLE as seen by the CPU. In this cycle, the MMU issues the physical address. At mid-cycle, the MMU issues the physical address strobe (PAS0) and data strobe indicating that a valid physical address is on the address bus and that data may be put on the data bus.

If the DTACK or SRDY signals are asserted in this cycle, the CPU will sample the data and terminate the transaction in the following cycle, otherwise the next cycle will be another WAIT-CYCLE.

4 If the DTACK or SRDY signals were asserted in cycle 3 and no bus exceptions occurred, then the data is sampled by the CPU in mid-cycle and the transaction is terminated. The DRDY0 signal is issued indicating that the transaction has completed without exceptions.

Write
Transaction
(See Figure 6)
Cycle Activity

1 Address, Status and Read/Write signals are generated at the beginning of the cycle.

In mid-cycle, the virtual address strobe is asserted.

2 Data is driven at the beginning of the cycle and the data strobe is asserted in mid-cycle.

If the address is a virtual address, it is tri-stated during this cycle.

If DTACK0 is received mid-cycle or SRDY0 is

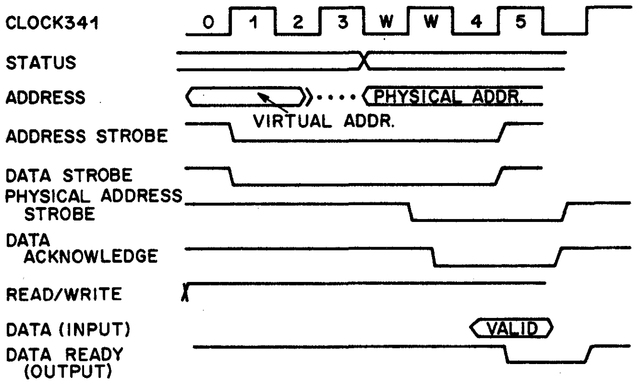


Figure 5—Read with address translation

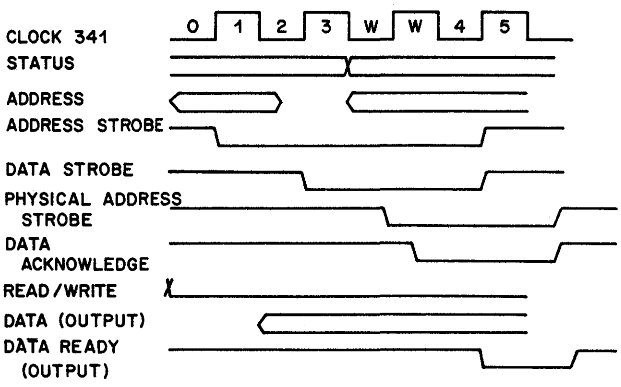


Figure 6—Write with address translation

TABLE I—Physical characteristics of the chip-set

Chip	No. of Transistors	Die Size	Total No.of I/Os (Active)
WE32100	180,000	1.0 cm^2	132/118
WE32101	92,000	0.7 cm^2	132/95
WE32103	13,000	0.4 cm^2	132/89
WE32104	113,000	1.0 cm^2	132/113
WE32105	4,500	0.4 cm^2	100/64
WE32106	160,000	1.0 cm^2	100/53

received at the end of the cycle, the access will be terminated during the next cycle. If no acknowledge is received, the next cycle will be a WAIT-CYCLE. Note that the MMU requires one WAIT-CYCLE to perform the virtual to physical translation.

3 Without an MMU, the transaction can be terminated if DTACK0 or SRDY0 was received during the previous cycle.

With an MMU, the physical address is issued, the physical address strobe is asserted at mid-cycle and the MMU's data strobe is asserted at the end of the cycle.

If DTACK0 or SRDY0 are received, the next cycle will terminate the transaction, otherwise, the next cycle will be another WAIT-CYCLE.

4 If the DTACK0 and SRDY0 signals were asserted the previous cycle, then the CPU will terminate the transactions by negating address strobe and data strobe. The DRDY0 signal will be issued if no bus exceptions were received.

Another salient feature of the chip-set is a preemptive method of obtaining the microprocessor's bus. Typical bus arbitration schemes do not allow a bus master to take control of the microprocessor's bus until the microprocessor has completed an ongoing transaction. The WE32100 CPU allows a non-interlocked transaction to be preempted by another bus master. After the new bus master returns control of the bus to the WE32100 CPU, the CPU will retry the preempted access. This preemptive bus arbitration, along with the common two-wire arbitration scheme allows easy implementation of multitiered system-bus structures. The preemptive arbitration allows for deadlock resolution in such multitiered systems.

PHYSICAL CHARACTERISTICS

Each VLSI chip in the WE32100 chip-set is state-of-the-art. Packaging is designed to maximize routability of chip I/Os as well as reduce $\delta i/\delta t$ noise. Table I summarizes the state-of-the-art characteristics of the chips in the chip-set.

SUMMARY

The WE32100 chip-set implements many minicomputer functions in six VLSI chips, with minicomputer performance.

ACKNOWLEDGMENTS

The synthesis of a VLSI chip-set such as the WE32100 chip-set requires contribution from many areas of expertise. The author wants to acknowledge the other chip-set architects: Thomas Lee, Peter Voldstad, William Dietrich, and Ulhas Gumaste. Also recognized are designers James Seery, Mark Kaplan, Frank LaRocca, Mark Thierbach, Jonathan Fields, Winston Pekrul, Tim Sippel, L. H. Blendinger, Barbara Tai and Lincoln Pierce.

REFERENCES

1. Goksel, A., J. Agraz-Guerena, H. Jacobs, J. Molinelli, P. Swartz, Y. Wo, and W. Troutman. "A Memory Management Unit For a Second Generation Microprocessor." Talk presented at COMPCON, Spring 1984.
2. Gumaste, U., P. Lu, and Z. Mao. "The Architectural Features of WE32106 Math Acceleration Unit (MAU)." Unpublished manuscript,
3. Goksel, A., J. Fields, U. Gumaste, and C. Kung. "An IEEE Standard Floating Point Chip." *Proceedings of the International Solid State Circuits Conference,* New York: IEEE, 1985.
4. Berenbaum, A., M. Condry, and P. Lu. "The Operating System and Language Support Features of the BELLMAC-32 Microprocessor." *Proceedings of Symposium on Architectural Support for Programming Languages and Operating Systems;* March 1–3, 1982; Palo Alto, CA.
5. Lu, P., W. Dietrich, M. Fuccio, L. Goyal, C. Chen, D. Blahut, J. Fields, A. Goksel, and F. LaRocca. "Architecture of a VLSI MAP for BELLMAC-32 Microprocessor." *Digest of Papers, COMPCON Spring 1983.* San Francisco, Calif.: IEEE, 1983.
6. Goksel, A., J. Fields, F. LaRocca, P. Lu, W. Troutman, and K. Wong. "A VLSI Memory Management Chip: Design Considerations and Experience." *IEEE Journal of Solid-State Circuits,* SC-19, No. 3 (1984).
7. *A Proposed Standard for Binary Floating-Point Arithmetic.* Draft 10.0 of IEEE Task, p. 754, December 2, 1982.

TRAC: An experience with a novel architectural prototype

by S. R. DESHPANDE, R. M. JENEVEIN, and G. J. LIPOVSKI
University of Texas at Austin
Austin, Texas

ABSTRACT

This paper presents a self-assessment of the Texas Reconfigurable Array Computer (TRAC) Project. Architectural decisions and implementation schemes are examined, and detail is focused on elements such as processor-to-memory design, along with the structure of the interconnection networks. Communication issues concerning circuit vs. packet switching and system synchronization are reviewed. Various test and debugging tools employed during the implementation phase are covered. Finally, now that TRAC is operational, overall positive and negative aspects of the design decision are summarized.

INTRODUCTION

The experience of building a prototype of the Texas Reconfigurable Array Computer (TRAC) has proved highly rewarding. It has improved our understanding of parallel architectures and, we hope, has made significant contributions to parallel processing.

TRAC employs the novel concepts of reconfigurability and space sharing in its organization. These are seen to be the key to the success of the general-purpose tightly coupled multiprocessors. Many other unique architectural features were included to enable it to perform equally well in both numeric and nonnumeric applications. Most of these features have fulfilled their promise; others have brought to light important issues that may demand further study.

This paper's two parts first assess the architectural aspects of the TRAC system and then describe some of the development tools used for the implementation.

PART I: ARCHITECTURAL ASSESSMENT

The Texas Reconfigurable Array Computer (TRAC) is an experimental array computer at the University of Texas at Austin. (The schematic layout of the system is shown in Figure 1.) It is expected to be a testbed for parallel algorithms and also a prototype for future general-purpose high-performance computers. All hardware-related development has been accomplished for Version 1.1, and all architectural functions have been implemented and tested. Work on the reliability of the system is being completed.

The architecture of TRAC is already well documented in several references.[1-3] So instead of describing the TRAC system again, this paper attempts to evaluate the merits and demerits of its many unique architectural and organizational elements. To analyze the TRAC architecture, we start by specifying the intended principles and goals for the TRAC project. The analysis will be easier in the light of these concepts.

TRAC Goals

1. Architectural Support: An architecture that aims to support a wide variety of models of parallel computations must satisfy the following requirements:
   a. Trivially, it must have an organization to accommodate a large number of processors.
   b. It must provide for different modes of communication between the processing units.
   c. It must have synchronization mechanisms general enough to allow an arbitrary combination of processors to be synchronized.
   d. It must be capable of SISD, SIMD, and MIMD modes of execution.[4] The system should be dynamically reconfigurable between tasks to support these modes of execution and to maximize the use of system resources.
2. Virtualize The Computation: The system must support vertical migration capability and make underlying hardware transparent to the user. A parallel architecture should provide a basis for implementing parallel languages.
3. Map The Architecture To The Algorithm: The system organization should be flexible enough to be able to mold the architecture to the algorithm, not the algorithm to the architecture, as has been the case in the past. It should make available to an algorithm the parallelism that it requests so that its true performance can be evaluated.
4. Give Attention To The Technology: The machine should be built modularly, with a minimum number of unique partitions. This would facilitate the translation of the design into the emerging VLSI technology. This last goal will allow us to assess the engineering decisions that went into the design of TRAC and to document our experience with the TRAC development effort.

The following sections review individual architectural and organizational elements of the TRAC system, outline the positive and negative aspects, and draw conclusions about whether their capabilities are desirable. In particular, the processor module, the memory module, and the banyan network

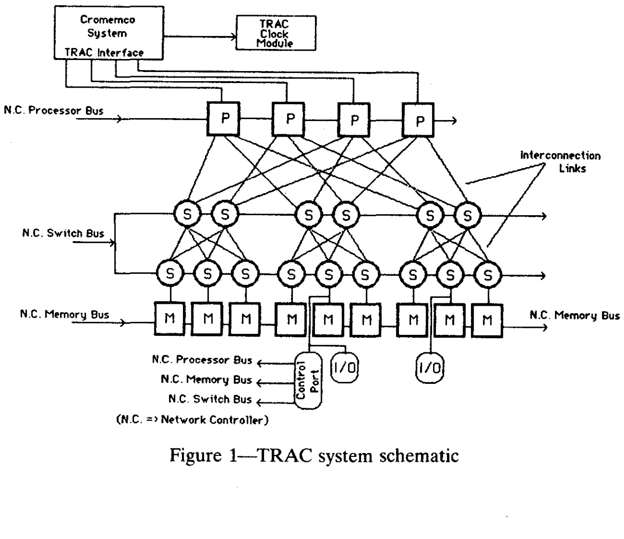


Figure 1—TRAC system schematic

form the major partitions. Instruction set is related to, and is therefore discussed along with, the processor module. The network structures and the dynamic reconfigurability of the TRAC system are analyzed in the section on banyan network. The synchronization and the interprocessor communication mechanisms are also discussed there.

Processor Module and Instruction Set

A very early design consideration was whether to use a commercially available microprocessor or to develop a customized processing element tailored to the special architectural specifications of the TRAC system. Several unique features warranted the development of an in-house processor. The following section briefly describes the processor module and details the salient features that determined the choice.

Processor module

The TRAC processor module is designed around the 2901 bit-slice microprocessors and uses the 2911s for microsequencing (Figure 2). Two 2901s are used to create an 8-bit-wide ALU with 17 temporary storage registers. Three 2911s handle the microsequencing and can address up to 8K words of control store, the least significant microaddress bit being determined by phase A or B of the system clock.

A TRAC processor module is a microprogrammable element with an 8-bit parallel arithmetic and logic capability. These processors themselves can become byte-slice elements to form a multibyte processing element by connecting them via a carry-look-ahead circuit embedded in the banyan interconnection network. The carry-look-ahead signals are controlled by microcode in the individual processing elements. The microprogrammable nature of the processor modules was essential to support these special capabilities, which facilitated the creation of a *varistructured* instruction set. Varistructure of the ALU provides a second dimension to the parallelism that already exists via the SIMD and MIMD architectures: Varistructure implies that the program code is independent of both the precision of the operand and the scalar/vector mode of the operation. The instruction execution is driven by the descriptor of an operand. The descriptor is stored in 2901 registers for fast access during instruction execution.

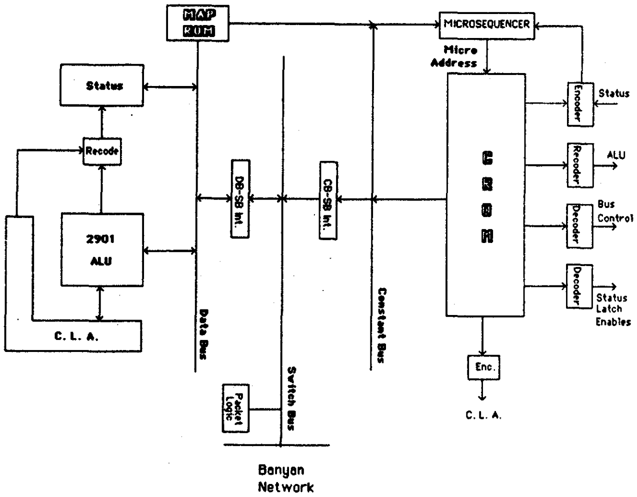


Figure 2—TRAC processor internal structure

In addition to the varistructured arithmetic, the microprogrammable design enabled the creation of a unique instruction set which was both efficient and tailored to a flexible array architecture. For example, all the branch instructions, when in an SIMD environment, automatically share conditions existing on all the processors belonging to a task and make a collective decision in a manner similar to ILLIAC IV. The following are typical branch instructions:

BAON—Branch if at least one of the processors contains a negative result.

BAC—Branch if all processors have their carry bit set.

The distribution of condition codes in a tightly coupled environment was made possible only through a microprogrammed processor.

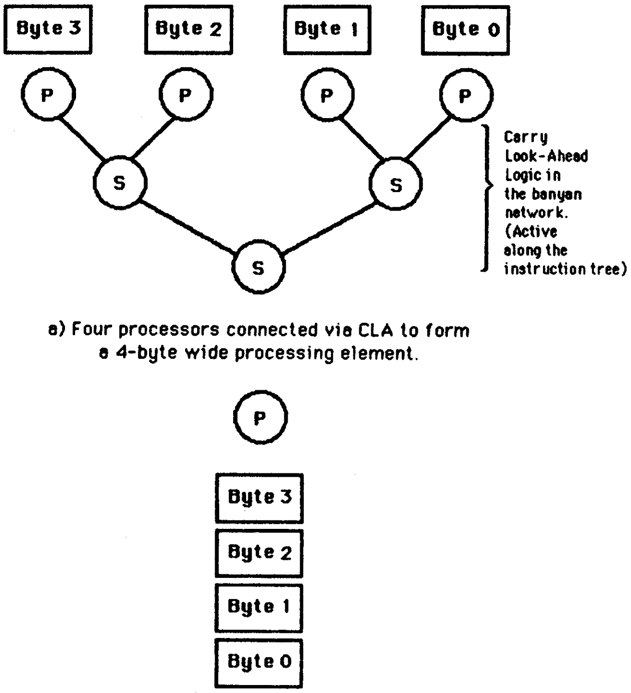
The TRAC processor module also includes a packet reception logic not available in commercial microprocessors. Of course, the same hardware could be interfaced to a microprocessor and read by software; but this would result in a lower performance, since many of the operations such as interrupt and exception handling required close microcode support.

In addition to the usual advantage of a flexible instruction set afforded by a custom microprogrammed implementation, one other factor played a role in favor of a custom-designed and microprogrammed processor module. Some independent considerations, to which we will refer later, dictated that the memory module would contain all addressing logic for the memory; the processors would be required to send down a command instead of an address to access an operand. The microprogrammed nature of the processor module would make it easy to incorporate this requirement.

The instruction set

The instruction set of TRAC is unique and is in keeping with the concept of virtualizing the processing element. All instructions are memory-to-memory and object-oriented. The instructions employ descriptor-based addressing of objects. Several issues are involved in the design of an instruction set for a reconfigurable architecture. Some of the important issues for TRAC were varistructurability, virtualization of the processor, complexity of instructions, support for high-level languages, and support for operating system functions. These are discussed below.

Varistructurability. The varistructurability itself was initially considered as an essential means of virtualizing the processors. Many popular word lengths are multiples of an 8-bit byte. Making the processors 8 bits wide and having a capability of cascading them via the carry-look-ahead linkages made it possible to create a processor of the desired word length. This also made the TRAC processor optimally suited for han-


Byte 3
Byte 2
Byte 1
Byte 0
P
P
P
P
S
S
S
Carry Look-Ahead Logic in the banyan network. (Active along the instruction tree)
P
Byte 3
Byte 2
Byte 1
Byte 0

a) Four processors connected via CLA to form a 4-byte wide processing element.

b) A single processor handles a 4-byte wide data element.

Figure 3—Varistructure

dling character data. Many processors can be connected to handle strings of characters as a single word (see Figure 3). The string operations are common in AI applications; therefore the TRAC architecture provides an attractive machine for implementing the AI algorithms. In general, varistructured instruction set for a processor is advantageous. The ability of a processing element to handle data of more than one word length eases the task of assembly language programming. Indeed, many commercial microprocessors available today already have this capability. Their instruction sets are orthogonal to the size of the datum; the size itself is specified as part of the opcode.

In contradiction to these expectations, the disadvantages of the generalized varistructure are found to outweigh its advantages. Although varistructurability is essential for the optimal use and performance of the hardware, its effectiveness depends on the way it is applied. Within TRAC, not only is each processor able to handle multiple precision data; but as described before, multiple processors can collectively operate on parts of the same multi-precision datum. This comes only with extra steps at the microprogram level to distribute the "carry" bits. The overhead is well justified for data of high precision. But in most applications the data are likely to be of a more or less standard size of up to 64 bits. For these sizes the overhead is simply not justified.

The generalized varistructure capability in TRAC also engenders another kind of inefficiency. All arithmetic and logic instructions in TRAC are vector–vector operations, and their implementations tacitly assume an SIMD operation. During multiplication, simple operations such as sensing the sign of multiplicand involves distributing that bit from one processor to all the others working on the same element of a vector. The implementation enforces all processors to stay in lockstep and not take advantage of the signs of individual elements to shorten the execution time. Consequently, although we insist that varistructure is an important concept for future generation of machines, we acknowledge that a very general scheme could lead to overheads in terms of processor synchronization.

Generalized varistructure also makes the task of the *loader* difficult. Depending on the number of processors scheduled for the task, it not only has to determine the placement of elements of a vector in the multidimensional memory, but also the placement of individual bytes within an element.

We considered earlier an argument in support of varistructure by citing an AI application. Studies conducted within the TRAC project have shown that varistructure across the processors is again not necessarily a good idea. If, instead, the processors within a task are allocated to handle different strings to be processed in an MIMD fashion, they may be expected to perform better. This is justified because a comparison between two strings often tends to fail early on, leading to a "wastage" of the processing elements working beyond the failing byte of the comparison. If one instead applies the parallelism vertically among one byte each of many strings, those that fail may be allocated to other strings, thereby improving the effectiveness of the parallelism.

Complexity of instructions and virtualization of processor. An assembly language programmer should not have to contend with the complexities of the hardware. In a system like TRAC there is an added complexity of network hardware and interprocessor synchronization. The TRAC instructions are implemented to make the underlying hardware completely transparent to the assembly language programmer. All structure-dependent instructions derive information from the structure descriptor, which is loaded in the processor registers. The instructions to load the structure descriptors are inserted at appropriate places by the system software. The final binding of the variables to their descriptors is performed by the loader.

Most operations that make use of the network are best handled at the microcode level; consequently, the policy that has been pursued is to create as powerful an instruction as possible without making it too restrictive. The TRAC assembly language reflects this choice. As mentioned above, all Conditional Branch instructions are vector-oriented. The branch is taken on conditions like "at least one true," "all true," etc. But when there is only one processor in a task, the instructions reduce automatically to conventional Conditional Branch instructions. The same holds for all arithmetic and logic operations.

Support for high-level languages and operating system. Many of the TRAC instructions are designed specifically to support high-level-language constructs. For example, the instructions Extract Byte/Element and Replace Byte/Element were created for assignment statements like A(3) = ⟨expression⟩, where only a single element of an array is modified. All instructions dealing with manipulation of data can handle vectors and therefore directly support vector operations in high-level languages. This is desirable because it leads to code optimization. Most instructions use descriptor-based addressing of objects and can easily be fashioned to handle capability-based access to system objects. There are also instructions to make use of the packet facility, both in user and supervisor environments.[5] There are two packet reception instructions: one is blocking while the other is non-blocking. The non-blocking instruction maintains a FIFO queue for the received packets.

One final note about the instruction set is in order. The subject of instruction set design has not been a primary topic of research in the TRAC project. Nonetheless, the experience of designing the current instruction set has already given us a valuable insight into the desirable capabilities of a tightly coupled multiprocessor architecture. In the future, the TRAC processors will carry experimental instruction sets for special applications.

Memory and I/O System

The memory and the input/output devices are connected to the base nodes of the banyan network. The TRAC architecture distinguishes between the two kinds of devices and has separate interfaces for both. The memory modules employ the Primary Memory Interface; the input/output devices use the Auxiliary Resource Interface. In this section the memory system is addressed first.

Memory system

A byte is the unit of data within TRAC. To accommodate that, each memory module is organized as four 16K × 1-byte frames (Figure 4). A memory module has all necessary hardware for mapping any one of its four frames anywhere in the address range. This provides a framework for implementing virtual memory system. Each module also houses pointer registers used for addressing storage. It has the necessary hardware for pointer manipulations such as post/pre, auto-increment/decrement, and add/subtract offsets. Processors send commands to the memory modules, where they are interpreted and executed in either the same or a subsequent memory cycle. The processors then take the necessary action either reading from or writing to the bus. In addition to these capabilities, the memory modules have hardware to initiate packets. They also incorporate user/supervisor protection and clean/dirty bits for their memory frames.

Memory modules in TRAC can operate in SISD as well as SIMD environments. They can form a multidimensional virtual memory system during SIMD processing. The memory design also makes it suitable for the varistructure architecture. The total addressable space of a processor is divided into four types: operand, data, control, and program. Two types, operand and data, are used for local and global storage of data. The operand and data spaces are as wide as the number of processors to allow for varistructure and hence for maximum parallelism during data processing operations. It should be noted, however, that to an individual processor these spaces still appear to be 1 byte wide. As will be seen later, the banyan interconnection network in TRAC supports a structure that allows the broadcast of information to a specified number of processors. This structure is used to distribute information from program and control spaces to the processors within a task. Since the program and control spaces store the common information as a single copy, they form single-byte-wide spaces. The program space stores instructions, and the control space stores descriptors for operands. This results in a reduction in the storage space necessary for a task.

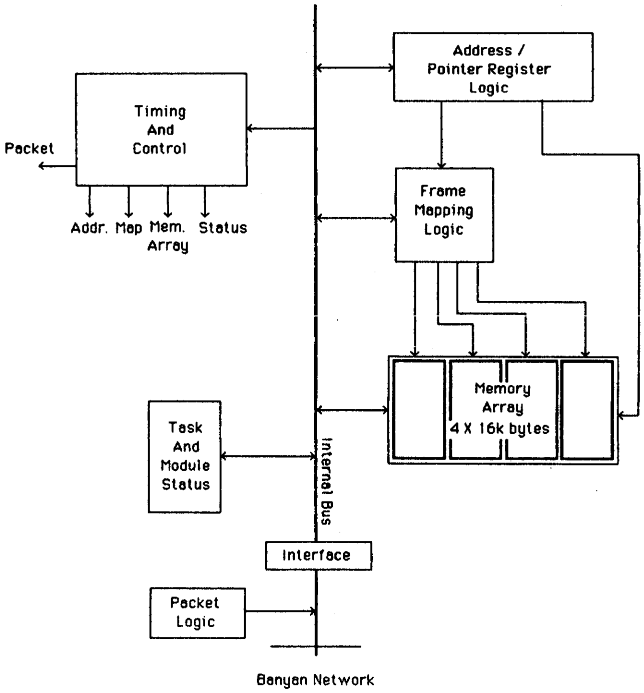


Figure 4—Memory module internal structure

A feature of the TRAC memory modules that has proved to be an excellent choice is the location of the memory address registers. As previously mentioned, they are located in the memory module. The advantage of this scheme has been twofold. First, the width of the bus traversing the switch has been limited to the width of the memory commands, which is typically smaller than the width of the memory address itself. Hence, the magnitude of control information transferred between the processor and the memory is minimized. This in turn reduces the number of connections on the switch and memory modules and the cost of the data path. The second advantage lies in the fact that commands other than those for loading the pointer values are independent of the size of the pointer registers and therefore the size of the memory. Consequently, the size of the address space can be changed with little change to the memory commands and no change to the processor hardware. The loading of pointers, over and above the operand fetch, takes at least three memory cycles. Thus for random accesses, the overhead of loading the pointers could be significant. But since the locality of reference of program behavior, along with array and string processing, in most cases dictates a more sequential pattern, this approach seems justified. Even if it were a problem, it could be rectified by providing additional registers in the memory module and by providing a capability to execute register arithmetic in the memory module under the control of a single command.

I/O system

The TRAC architecture has a well-defined interface between the interconnection network node and the primary memory module, called the *Primary Memory Interface.* All other secondary memory systems and I/O are implemented via an interface called the *Auxiliary Resource Interface* (ARI). The ARI access has been made device-independent at the user level. The transfers between an Auxiliary Resource (I/O device) and the primary memory are implemented via descriptor-based instructions. The descriptors have the same general format, although their contents are specific to the device being accessed. The calling sequences of the instructions are independent of the device being addressed, making the hardware details of the underlying device transparent to the user. Thus the concept of ARI has become central to the virtualization of I/O in TRAC. (The concept is not dissimilar to that of the /dev file in UNIX.) Also, the actual transfers of data between the device and the primary memory are complementary, allowing transfer of data during every memory cycle. The ARI concept has already been used to connect devices such as terminal,[6] printer, disk, self-managed-secondary-memory,[7] and the control port[6] to TRAC.

Banyan Interconnection Network

The successful implementation of the banyan interconnection network can be considered to be the most important contribution of the TRAC project so far. It is a two-sided, multistage network with processors at the apex end and memories or input/output devices at the base. It has been built modularly with unique partitioning properties; it has been built with a single building block called the switch module. The switch module itself is easily segmentable and amenable to VLSI implementation. Furthermore, a study[8] has shown that it is feasible to implement the interconnections within the banyan network by using light pipes.

The primary purpose of a close-coupled computer network is to provide mechanisms for processor–memory and processor–processor communication and those for interprocessor synchronization. The performance of algorithms is directly related to the effectiveness of these mechanisms. Most of the unique characteristics of the TRAC architecture result from the capabilities of the supporting interconnection network. To begin with, its partitioning allows multiple independent tasks to run simultaneously in the system without being aware of the other tasks. It supports multiple SIMD (or SISD) and space-shared processing. This is an important property for the future generation of multiprocessors, which will employ a large number of processors, only a subset of which will be dedicated to a single task. The network supports both packet- and circuit-switched modes of data movement. The packets are essential for implementing an arbitrary permutation on a blocking network while also furnishing asynchronous communication between the processing elements. The circuit-switched modes of interprocessor communication occur in TRAC in the form of *shared* and *instruction trees.* It is believed that the presence of both the circuit-switched and the packet-switched modes of communication is necessary to produce the best performance. The mechanisms of both types of communication in TRAC are described below, and a brief discussion is presented in support of the need for both modes to coexist.

Packet-switched communication

The packet switching mode of communication affords an asynchronous mechanism of communication between the TRAC processors. The communication channel thus created is guaranteed to be present regardless of the other network structures. Although a packet may be delayed because of congestion in the network, it is not indefinitely blocked. This facility therefore provides an ideal medium for implementing a task-to-task message-passing mechanism. For example, in TRAC, it allows a direct implementation of a high-level construct called the *channel* in the Computational Structures Language (CSL),[9] a job control language designed for reconfigurable architectures.

Circuit-switched communication

The banyan network in TRAC supports a dynamic generation of three tree-shaped circuit-switched communication structures: shared tree, data tree, and instruction tree. Although these structures possess a tree topology in the interconnection network, they electrically exhibit bus properties. The structures are created via logic circuits distributed over the banyan network, which are collectively termed the Network Controller.[10] The network controller hardware is highly parallel and is capable of establishing or deleting a structure once every few memory cycles. The operation and architectural advantages of the three network structures are discussed below.

Shared tree. The banyan interconnection network supports construction of a tree structure, called a shared tree, with a memory (*shared memory*) at its root and processors at its leaves (Figure 5). The shared tree implements a priority scheme among contending processors trying to "acquire" the shared memory. Only one leaf node may be actively connected (in what is called an active path) to the root at any given time. A hardware semaphore is hence enforced via this mechanism, thus preventing simultaneous updates of the contents of the shared storage. Once an active path is established, it remains so until the processor owning it "releases" the memory module, at which time the shared tree is again available to be acquired. The importance of shared memory arises from the fact that after being acquired it acts simply like an extension of the processor's private memory and is available to it without contention. Thus, one processor may acquire a shared memory, load it with data, and release it. Then another processor can acquire that shared memory and operate on the data as though from its own *private* memory. This pro-

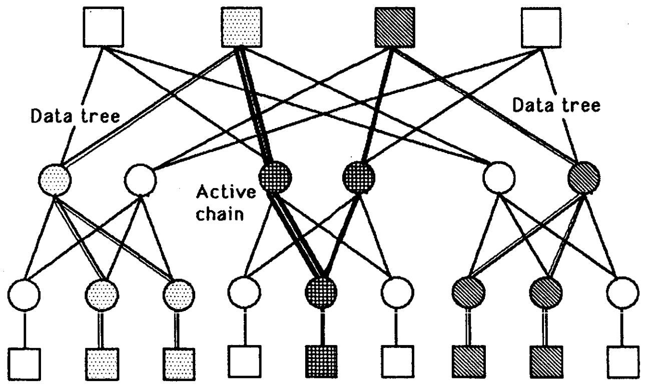
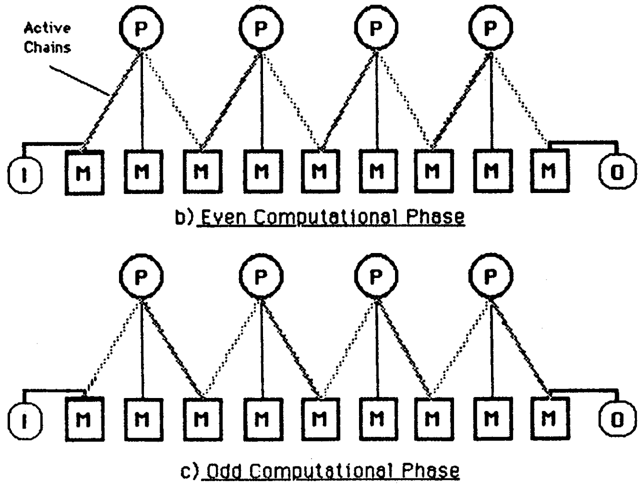


Figure 5—Share tree

vides a natural means of creating macropipelines of computations (Figure 6).

At a clock rate of 1 MHz, it takes on the order of 20 microseconds for an acquire and release combination operation. The current size of the memory module is 64K bytes. This translates to an effective transfer rate of 3.2 gigabytes/sec. Conceptually, then, the shared memory provides a means to achieve very high bandwidth channels. As more and more algorithms are cast in parallel form, we envision a rising need to decrease the interprocessor communication cost and therefore an increasing demand for such high-bandwidth channels. It is felt that packet switching alone is not sufficient for a general-purpose machine like TRAC. This belief is held because packets inherently have high latency. Granted that the transmission of packets can be pipelined to achieve low net delays, the packets may still suffer delays as a result of "collisions." An important premise of parallel computing is that, for an algorithm to be effective, its interprocessor communication overheads should be minimized and the host architecture should provide mechanisms offering a high interprocessor communication bandwidth. In general, the packet mechanisms do not offer this. Thus, although packet switching is a universal communication mechanism, it is not sufficient by itself. The circuit-switched mechanisms like the shared tree ought to be employed to improve the performance.



a) Initial Setup

b) Even Computational Phase

c) Odd Computational Phase

Figure 6—Macro pipelines

The shared memory mechanism in TRAC is not as universal as it may appear. It depends on the creation of a shared tree structure in the network and is therefore subject to blockages. But since any memory in the network can work as a shared tree root, we anticipate a low likelihood of failure to generate a shared tree over a given set of processors. This has been verified by simulation studies.[11]

The concept of shared memories has helped us develop a programming methodology for parallel machines. It is believed that the sharing of a common storage area by contention will lead to inefficiencies in processing when a large number of processing elements are involved; sharing through cooperation is instead suggested. As some of our studies have shown, one can cast parallel algorithms into phases of computation.[12] These phases are SIMD (or SISD) in nature, and do not involve sharing of information. Sharing, in general, takes place at the boundaries of computational phases and is explicit and predefined. Thus an entire algorithm can be encoded as a number of computational phases interspersed with steps at which the processes cooperate to share data (Figure 7). A recent study of some numerical and image-processing algorithms has shown this to be a reasonable approach.[13] Thus if the sharing of data can be structured and executed explicitly, a shared-memory mechanism can be used to transfer data between the processes at a very high rate. This would eliminate the overhead of contention almost completely.

Data tree. The shared tree, as explained above, is a means of implementing the common data storage to be shared by many processors. A data tree, on the other hand, is a structure for allocating "private" memory to an individual processor. The concept is similar to local memory as opposed to global memory in traditional architectures. In TRAC there is a strict distinction between the private storage of a processor and the storage that it may share with others. The separation is physical and is created at the beginning of a task.

The data tree is a tree structure with the root at a processor module and leaves at the memory modules (Figure 8). Since the banyan network ensures a path from each processor to every memory module, any number of memory modules may be attached to form a data tree. The data tree concept provides a flexible method of tailoring the size of private storage of a processor to the requirement of local addressing space. Like the shared tree, a data tree is set up, and the processor and the associated memories form a bus-oriented system. The structure is electrically isolated from other network structures, and it does not suffer from any contention problems.

A data tree is the circuit-switched equivalent of bidirec-

```
job   parallel_sort;
var    i, j, num_mems, num_procs  :  integer
begin
   construct
      tasks sort(i) : sortfile[s(i)]  range  i = 1 to 8 ;
{create 8 sequential sorting tasks using file "sortfile"}
   end;

   execute   sort(i)   range  i = 1 to 8;
{execute the 8 sorting tasks in parallel}
{the next phases are to merge their results}
{recursive merge algorithm is applied to allow parallelism}
   num_procs = 4;
   num_mems = 2;
   repeat

      cobegin
        construct
           tasks   merge(i)  :  mergefile[s(num_mems*i - j)]
                                        range  j = 1 to num_mems;
                                                 i = 1 to num_procs;
        end;
        // ( with  s(num_mems*i - j)   do   execute merge(i) )
                                           range  j = 0 to num_mems-1;
                                                    i = 1 to num_procs;

      coend;

      num_procs = num_procs DIV 2;
      num_mems = num_mems*2;

   until  (num_procs = 0);
end.

{The above CSL program implements the well-known but}
{simple parallel sorting algorithm.  8 independent tasks}
{are initially executed in parallel.  The 8 tasks share the}
{array to be sorted : "s".  "s" is distributed over num_mems}
{memories.  After the parallel sort phase is over, the}
{merge tasks are scheduled.  They recursively combine the}
{result of the privious phases to obtain the sorted array.}
```

Figure 7—A CSL program

tional communication between a processor and its private memories. The packets, on the other hand, provide a mechanism of communication between data trees, as do shared trees mentioned above.

The concept of shared memory with explicit handoff results in a unique architectural advantage: ease of incorporation of cache memory. Although the TRAC processors do not have any cache memory in them, it can be easily incorporated. An acquired shared memory becomes an extension of the private memory of a processor. No other processor is able to access that storage space during that time. The owner processor can bring the shared data into its cache and freely update it without risking the traditional problem of cache consistency. The cache can be flushed out just before releasing the shared memory.

Instruction Tree. In an SIMD task, an instruction or a common data item is broadcast simultaneously to all processors in the task. In TRAC, the network structure provided to support this broadcast is the instruction tree (Figure 9). Under the control of the processors, a byte is broadcast from a single source to many destination processors. In addition to the instruction opcode, the structure is also used to broadcast control information, such as descriptors and subroutine addresses. It can be used to distribute and collect data from a task's processors. A byte is transmitted over the instruction tree in a circuit-switched fashion. This necessitates that the processors be in lockstep, and the transfer takes one memory cycle. The instruction tree proves to be a valuable mechanism for intratask communication, since the processors are already in lockstep synchronization.

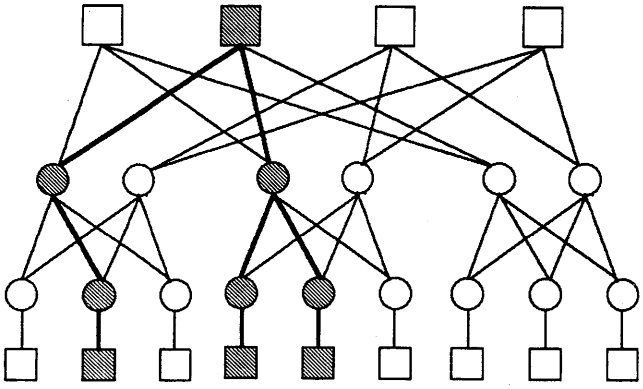


Figure 8—Data tree

Carry-look-ahead logic (CLA)

An element of carry-look-ahead logic is embedded in each switch module. The logic accepts the block-generate and block-propagate inputs from two links toward the processors and sends a group-generate and group-propagate to the level above (toward the memories). It receives the group-carry input from the higher level and distributes the block-carries to the level below (toward the processors). In TRAC, the CLA is activated over an instruction tree path. The detailed operation of the CLA circuits can be found in Reference 14.

The use of CLA for arithmetic is obvious, but it is time-multiplexed to act as a one-bit-wide communication mechanism and priority encoder as well. For example, branch instructions assert the block-propagates on individual processors to achieve broadcast of a single bit to all the processors in the task.

The CLA is active along an instruction tree during a certain phase of the system clock, and is active along a shared tree

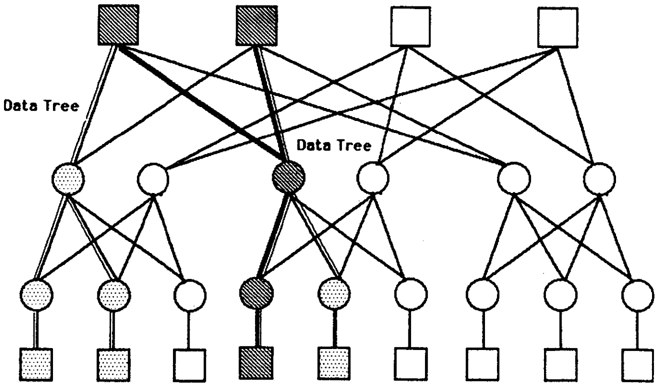


Figure 9—Instruction tree

during another phase. During the latter it acts as a priority circuit, relinquishing control to only one processor at a time. A more complete discussion of the use of the CLA for processor synchronization has been reported earlier.[5] All the synchronization mechanisms in TRAC provide one-to-many type of synchronization, and it is difficult to synchronize an arbitrary pair of processors without prior planning. A more general synchronization scheme may be derived after studying the needs of different algorithms. It has been realized that the synchronization primitives are essential in a system like TRAC, and the CLA in TRAC is a step in that direction.

The Availability Goals

Two important engineering goals, reliability and testability, were missing from the original charter on account of the lower priority assigned to them as research issues for TRAC Version 1.1. For this reason, only a few experiences have been cited, and recommendations made. An excellent reference to reliability techniques may be found in Reference 15.

A machine of the complexity of TRAC, because of its connection-intensive organization, is very prone to failures, such as "opens" at a cable connection point. The basic reliability features, such as parity detect or correct, are essential for network-oriented systems.

In TRAC the connections between the processors and the memories are made through the switch modules. When a word is fetched from a memory, a single bit error may occur anywhere in the path—from our experience, probably at the point of connection of the cables. Presently, there is no hardware in the TRAC system to detect an occurrence of such a fault on-line. The TRAC processors should be able to detect parity errors for both the data they receive and the data they send. It should also be able to distinguish between the two. A choice has to be made between providing parity bits for storage elements of the memory modules and providing a pair consisting of a parity checker and a generator at the module interface, or both.

TRAC was designed with a space-sharing rather than a time-sharing programming discipline in mind. This implies that the interconnections between the processors and the memories be established at the beginning of a process and remain intact until the end of the process. During this entire duration only the designated process executes on the processors. The context switch takes place only between the executing process and the operating system elements servicing the calls to the privileged routines and exceptions. Thus if a parity fault occurs for even one processor, the entire process tends to get halted. This can be a major overhead for processes in case of transient errors. In the light of this discussion, the Single Error Correction-Double Error Detection seems to be a better strategy than the Single Error Detection, in spite of its higher cost. Extensive retry and recovery mechanisms could also be employed.[16] In any case, the need for on-line error detection and correction mechanisms in multiprocessors can hardly be overstated.

The banyan network allows the testing of switch modules so that a faulty switch module may be isolated.[12] To achieve this, a background process can execute a test continuously for a failure to occur. On occurrence and isolation of a failure, one switch module or a pair of switch modules can be flagged as unusable. Thus, in the future schedule of processes, all the paths using the faulty modules can be avoided. Under these circumstances, the system will gracefully degrade in performance. Both processor and memory modules have network status registers containing fault bits so that, should a fault be detected, the fault bit is set and that component of the system is no longer allocated by the network controller.

Testability has also lacked attention in the TRAC system. The swtich modules are most amenable to design for testability. They are essentially combinational and contain very few flip-flops. Well-known techniques such as the LSSD or the Scan Path can be employed at little extra cost to improve testability.[17] The testability of processor boards can be improved by adding a few registers that can be externally read from and written into. The TRAC clock module already allows externally controlled stepping. The addition of an externally loadable micro-instruction register would allow the testing of the microsequencers and the basic ALU operations. Further, the addition of a small amount of writable control store can facilitate on-line diagnostics and testing of new microroutines.

PART II: DEVELOPMENT AIDS

A number of tools were generated in house to help develop hardware and microcode for TRAC and were used in addition to the traditional tools such as the high-bandwidth oscilloscopes and logic analyzers. These tools were built around a Z-80-based Cromemco microcomputer. The tools included monitors, testers, and microprogram development aids. These are discussed in more detail in the following paragraphs.

Monitor Programs

The TRAC system is designed around a central system clock supplied to the boards of the system via a tree-shaped distribution system. The basic clock is a 50-MHz signal, and it is used to derive the multiphase TRAC system clock. The stepping of the system clock can be controlled either externally or internally. In the latter mode the system operates under a free-running clock. In the former mode the system can be held in a particular clock phase indefinitely, thus facilitating manual probing and debugging of the circuit.

Two monitor programs, MOE and RABUG, were written for the Cromemco system to help develop the hardware and to facilitate debugging of the microcode. A parallel interface was built that allows the Cromemco microcomputer to read and write to the network interface busses and the microaddress busses of all the processor modules (Figure 1). Through separate interface, the microcomputer is able to control the clock. By proper use of these interfaces, the monitor programs running on the Cromemco can step through the phases, microcycles, and memory cycles, and in addition can read data from or write data to the busses in the TRAC system.

Monitor program MOE can sequence through a specified number of phases, microcycles, machine cycles, and TRAC instructions and can display or print the data from the busses and display the contents of the memory pointer registers and the processor status registers for individual processors. To facilitate this, instrumentation was added at the microprogram level to output the required information on the network bus. At the beginning of each instruction, a microprogram routine is executed, which supplies the processor status information and memory pointer information to the monitor program, which then displays it on the CRT screen. The monitor program is also capable of receiving its commands from a batch file. The batch files are created to run the machine through an entire program and list bus data or processor status any number of times. This way it is possible to exercise the machine for long periods of time and capture faults if they occur.

Monitor program MOE, mentioned above, explicitly controlled the clock, executing considerable Z-80 code for each TRAC clock step. As a result, the TRAC hardware was exercised only at slow speeds. After all functionalities of the architecture were developed and tested, a need was seen for a more sophisticated monitor, which would run the system at the rated speeds of 10 kHz, 100 kHz, or 1 MHz and still retain the debugging capability. Coupling logic was added to enable the TRAC processors and the Cromemco system to handshake and to allow the monitor program RABUG to switch between free-running and controlled stepping of the system clock. A capability was also added so that the monitor could send commands, in addition to data, to the memory modules. This latter capability has proved to be helpful for software debugging, since via the monitor the contents of the memory can be inspected and/or modified. It is now also possible to insert *break points* in the programs to further facilitate their testing.

Test Programs for Hardware

The subject of testing goes hand in hand with the subject of digital system development. Usually automatic test equipment is employed in a manufacturing environment to automate and to speed up the testing of subsystems. However, the capability of automatically testing modules was found desirable even during TRAC system development because of the replication of circuitry in the submodules into which the system is partitioned. The availability of commercial automatic testers was financially infeasible; therefore in-house aids were developed to test the switch and memory modules.

The switch modules have predominantly combinational circuitry and are easily adapted to input-output specifications. A table-driven test system was developed around the Cromemco microcomputer to inject test patterns into the switch modules via the interconnection links. It reads a vector from the table, sequences through the clock if necessary, and checks the module output against the expected pattern. If a mismatch occurs, the system indicates this by displaying the bit positions of the output vector at the time the error occurred. Since the test software was executed on the Cromemco microcomputer, the tests could be conducted only at slow rates; but this was satisfactory because only the logic was tested with this software. A similar tester was developed for the memory and the input/output modules. The memory modules were tested only for their support circuitry; the memory array was tested in the system by executing TRAC software. The network controller hardware was tested by a program and special interfaces to the TRAC system that allowed complete control of the reconfiguration hardware.

Microprogram Development Support

The choice of custom-designed microprogrammable processor modules led to generation of a complete set of microprogram development tools. The microprogram development tools include an assembler that executes on the Cromemco microcomputer and translates symbolic microroutines into the bit patterns. In addition to the assembler, the development system includes programs for linking the microroutines, encoding the raw patterns into PROM images, and programming the PROMs. All these programs execute on the Cromemco system.

Software Development Aids

A software simulator for TRAC was written to help develop system and application software while the hardware was under construction. The software simulator is able to provide the parallel programming environment available on TRAC. It is also able to simulate the shared-tree concept and the packet communication. A Pascal compiler, an assembler, and a loader have also been developed for the TRAC system. These programs can also generate code that can be interpreted by the TRAC simulator.

SUMMARY

The TRAC system has many unique architectural features based on the supporting banyan interconnection network. These mechanisms are meant to support the parallel processing constructs. In particular, the shared tree is found to be a powerful mechanism to implement a high-bandwidth communication channel. The data tree is a network means of partitioning the overall system storage area physically and incrementally. Since a single processor accesses the data tree, it is considered to be an effective method of implementing local storage.

One of the important elements of a parallel architecture is the synchronization mechanism. The CLA-linkage is used for both synchronization and communication. Intratask communication can also be achieved via the instruction tree.

The processor and the memory modules are designed to serve the architectural goals of varistructurability and virtualization of computation. The goal of varistructurability has not proved to be as profitable as was expected. The goals of reliability and testability were left out in the initial list. These goals are important for a multiprocessor system, since the likelihood of failure is higher as a consequence of increased hardware complexity. The packet communication facility has provided a means for intratask data permutation, intertask communication, and operating system message interface.

In perspective, the TRAC project has been successful in meeting most of its architectural goals and has given insights useful for the next generation of parallel computers. The study of the TRAC system has also exposed new areas of research, such as parallel I/O and the modelling of parallel computations on reconfigurable machines.

REFERENCES

1. Kapur, R. N., "Organization of the TRAC processor-memory subsystem." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 623–629.
2. Sejnowski, M. C., E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski. "An Overview of the Texas Reconfigurable Array Computer." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 631–641.
3. Premkumar, U. V., R. Kapur, M. Malek, G. J. Lipovski, and P. Horne. "Design and Implementation of the Banyan Interconnection Network in TRAC." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 643–653.
4. Flynn, M. "Some Computer Organizations and Their Effectiveness." *IEEE Transactions on Computers,* C-21-9 (1972), pp. 948–960.
5. Rathi, B. D., S. R. Deshpande, M. C. Sejnowski, D. Walker, R. M. Jenevein, G. J. Lipovski, and J. C. Browne. "Specifications and Implementation of an Integrated Packet Communication Facility for an Array Computer." *Proceedings of the 12th International Conference on Parallel Processing,* Silver Spring, Md.: IEEE Computer Society Press, 1983, pp. 51–58.
6. Prakash, A. "Design and Implementation of an Input/Output Interface for TRAC." M.S. Thesis, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas.
7. Rathi, B. D. *Principles of Operation of TRAC's Self-Managing-Secondary Memory.* TRAC Report, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas, 1981.
8. Goyal, A., and G. J. Lipovski. "Light Pipe Implementation of Banyan Networks." *Journal of Digital Systems,* 6-4 (1982), pp. 367–386.
9. Browne, J. C., A. Tripathi, S. Fedak, A. Adiga, and R. N. Kapur. "A Language for Specification and Programming of Reconfigurable Parallel Computation Structures." *Proceedings of the 11th International Conference on Parallel Processing,* 1982, pp. 142–149.
10. Jenevein, R. M., and J. C. Browne. "A Control Processor for a Reconfigurable Array Computer." *Proceedings of the 9th Symposium on Computer Architecture,* Silver Spring, Md.: IEEE Computer Society Press, 1982, pp. 81–89.
11. DeGroot, R. D. "Mapping Computation Structures onto SW Banyan Networks." Ph.D. Dissertation, Department of Electrical and Computer Engineering and Department of Computer Sciences, University of Texas, Austin, Texas.
12. Kapur, R. N. and J. C. Browne. "Block Tridiagonal System Solution on Reconfigurable Array Computers." *Proceedings of the 10th International Conference on Parallel Processing,* Silver Spring, Md.: IEEE Computer Society Press, 1981, pp. 92–99.
13. Yesrebi, M., S. Deshpande, and J. C. Browne. "A Comparison of Circuit Switching and Packet Switching for Data Transfer in Two Simple Image Processing Algorithms." *Proceedings of the 12th International Conference on Parallel Processing,* Silver Spring, Md.: IEEE Computer Society Press, 1983, pp. 25–28.
14. Lipovski, G. J. "An Organization for Optical Linkages Between Integrated Circuits." *AFIPS, Proceedings of the National Computer Conference* (Vol. 46), 1977, pp. 227–236.
15. Siewiorek, D. P. and R. S. Swarz. *Theory and Practice of Reliable System Design.* Bedford, Mass.: Digital Press, 1982.
16. Deshpande, S. R. *Hardware Retry Mechanism for TRAC.* Project Report, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas, 1984.
17. Williams, T. W. and K. P. Parker. "Design for Testability." *IEEE Transactions on Computers,* C-31-1 (1982), pp. 2–15.

SUGGESTED READINGS

1. Opper, E. "Fault Diagnosis of Banyan Network." Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas, 1984.
2. Deshpande, S. R. *Design and Implementation of the Control Port for TRAC.* TRAC Report, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas, 1983.
3. Sejnowski, M. C. *RABUG User Guide.* TRAC Report, Department of Electrical and Computer Engineering, University of Texas, Austin, Texas, 1984.

Hypercube architectures

by LYLE L. WELTY and PETER C. PATTON
Microelectronics and Computer Technology Corporation
Austin, Texas

ABSTRACT

The state of the art of hypercube architectures for parallel processing is represented today by the Cosmic Cube and based on a binary n-cube interconnection architecture developed by Seitz and Fox at the California Institute of Technology. As a means of achieving performance-oriented parallel processing for computationally intensive problems from the natural sciences which enjoy the property of physical (or logical) partitionability, the Cosmic Cube represents a proven architecture which can be implemented in a cost-effective manner using any microprocessor with a floating-point coprocessor. We may expect to see commercial versions of the Cosmic Cube based on 16- and 32-bit microprocessor chip sets now entering the market. The description of Cal Tech Concurrent Computation Project (C^3P) presented here was prepared as part of an MCC catalog of parallel processing architecture development projects.

INTRODUCTION

The availability of high-performance 16- and 32-bit microprocessors such as the Intel 80286 and 80386, Motorola 68010 and 68020, and NS 16032 and 32032 together with their floating point coprocessors suggests the possibility of building very high-performance computers from high-volume, low-cost components rather than low-volume, high-cost components.[1] In fact, there is a rather broad consensus among computer engineers that parallel processors based on VLSI microprocessors represent the solution to the performance problem in computing. This consensus breaks down as soon as one asks how we ought to organize N microprocessors to do a given problem O times faster than it can be done on a single processor, or how to do a problem P times larger on Q machines in the same time the given problem runs on a single processor. There are nearly 100 parallel processor architecture research and development programs underway today, but the Concurrent Computation Project at the California Institute of Technology is one of the furthest ahead in proving the capability of concurrent VLSI architectures for computationally intensive scientific problems. It is also the nearest to commercial exploitation.[1–5]

A problem in parallel processing research today is an overabundance of hypotheses and attempts to discover a new interconnection architecture for N processors; there are already many more such suggestions than can be reasonably evaluated. The fundamental research problem is one of software. The solution begins with the selection of a promising application and the known algorithms for solving that problem on a sequential processor. One must generally develop new *parallel algorithms* based on a parallel model of computation rather than a sequential one.[4–8] The language issue arises when one must either extend a prescriptive language like FORTRAN or C to allow the expression of manually extracted parallelism, or adopt a descriptive language like FP, SASL, or PROLOG to allow the automatic extraction of parallelism. The former represents a technology that is mature and ready for commercial exploitation; the latter is a new technology that, while promising, may be five or more years away from efficient application in commercial systems.

The CalTech Concurrent Computation Project (C^3P) employs extended FORTRAN and a sophisticated operating system together with a programming style that significantly aids the scientist end-user in the manual extraction and expression of parallelism for a broad class of physical, numerical, and signal processing problems.[4,5,9] Results to date have exhibited significant performance gains with very high cost-effectiveness on a 6-dimensional binary hypercube architecture employing an Intel 8086/8087 at each vertex.[5,8–11] It is clear that the hypercube architecture will scale upward to systems with more powerful vertices (say Intel 80287/310 or Motorola 68020 chip sets) and to higher-dimensional cubes (say 10 or 12 dimensions) for the applications upon which it has been shown to be effective. Whether an architecture which employs message passing interprocess communication exclusively will extend to the symbolic applications considered typical of the so-called fifth generation, and exhibit similar scalability, is still a research issue.

ANTECEDENTS OF THE PROJECT

The possibility of interconnecting a large number of small floating-point computers in regular or homogenous 2-, 3- or higher-dimensional arrays to solve very large problems in scientific computation has been considered for more than two decades.[12] The current availability of powerful microprocessor chip sets including floating-point coprocessors has promoted the binary n-cube or hypercube architecture to physical realizability. In the mid-seventies, M. C. Pease proposed a binary n-cube architecture for several numerical algorithms including the fast Fourier transform;[13] in 1975, IMS Associates announced *The Hypercube,* a 4-dimensional computer based on his ideas.[14] The Hypercube had thirty-two 8-bit microprocessors arranged in a 4-cube with two processors at each node, one for computation and one to handle internode communication. The machine was not commercially successful because its architecture was too advanced for the microprocessor technology available to implement it. The binary hypercube architecture project at CalTech is based on extensive program modeling and simulations carried out between 1980 and 1982 by Charles R. Lang.[15] It was from this work that the communication plan of a binary n-cube, the bit rates of the communication channels, and the organization of the operating system primitives were chosen. The Cosmic Cube[16] implementation was developed as a collaboration between computer science (C. Seitz) and high energy physics (G. Fox). The major goal of the project is the development of concurrent algorithms, and implementing and running them on parallel processors to solve significant scientific problems.[1,5,9] The Cosmic Cube development project is the current flag-carrier for the hypercube architecture concept; however, we expect to see other implementations of this architecture appear during 1985.

In the 1982/83 timeframe, C^3P developed the Mark I machines based on the INTEL 8086/8087microprocessor set with each node having 128 Kbytes of RAM and six communication channels. Both 8- and 64-node machines of this design have been run successfully since October 1983. The 64-node Mark

I machine running at 5Mhz has a power claimed to be roughly equivalent to eight DEC VAX 11/780s for efficient implementations of appropriate applications.

During 1984, CalTech developed (with the Jet Propulsion Laboratory doing the actual construction) a new set of machines (the Mark II series) based on the Mark I design but with twice the memory per node and eight channels each. This allowed a 128-node (seven dimension) configuration with a spare channel on each node for data communication to external devices such as disks and an Ethernet-based network.

Most of the current literature available does not refer to these machines as Mark I or II, but as Nearest Neighbor Concurrent Processor (NNCP), Hypercube (boolean hypercube topology), Homogeneous (of uniform structure) machine, or the Cosmic Cube, with the NNCP being the generic descriptor.[1-3,5,9]

TIME-FRAME OF THE C^{3}P PROJECT

The Mark I logical design was completed in the Fall of 1981. A 4-node (or 2-dimensional cube) prototype was implemented using wire-wrap boards in the Spring of 1982. This prototype was operational by July 1982. A 6-dimensional 64-processor extended version became operational in October, 1983. The 64-processor Cosmic Cube was employed for the solution of many large scale physical and engineering demonstration problems that have been reported and published.[10,11,17-19] A users manual was published to aid pioneering users in concurrent computation on the Cosmic Cube in August 1984.[20]

GENERAL ARCHITECTURAL BLOCK DIAGRAMS

The CalTech concurrent processor has 64 nodes arranged in a Boolean or binary hypercube architecture; this is, in general, 2^p computers with the same connection topology as the vertices of a cube in p dimensions. For the 64-node machine, $p = 6$. Figures 1–3 are for 3- and 4-dimensional cubes which are smaller versions of the communication scheme used in the 6-dimensional, 64-node Cosmic Cube.

DESCRIPTION OF THE GENERAL ARCHITECTURAL CONCEPTS

General

The Cosmic Cube is a concurrent VLSI processor having 64 microprocessor nodes arranged in a binary hypercube architecture. Each node is a single computer based on the Intel 8086/8087 microprocessor chip set containing 128 KB of RAM memory on the processor board. The machine has an MIMD architecture. Each node runs asynchronously with no shared memory and is connected to its six neighbors in the hypercube topology by four 64-bit wide parallel data paths connecting 64-bit wide FIFO buffers. Communication data rate is about four Mbits/sec between directly connected or nearest-neighbor nodes.

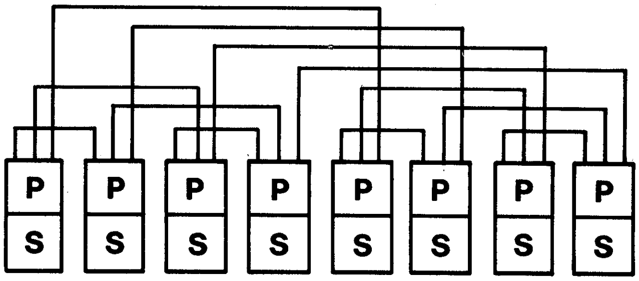


Figure 1—Interconnection of a three-dimensional cube message passing architecture

Design

A binary n-cube communication scheme was chosen because this network was shown by simulation to provide very good message flow properties in irregular computations. The n-cube that is used to connect $2^n = N$-nodes is assembled from two $(n - 1)$-cubes, with corresponding nodes connected by an additional channel (see Figure 2). This property simplifies the packaging of machines of varying size and explains some of the excellent message-flow performance properties of the binary n-cube on irregular problems.

Channel speed is fairly slow when compared with the instruction rate. The communication channels are asynchronous, full-duplex, and include queue storage for a 64-bit hardware packet both in the sender and receiver in each direction in order to decouple the sending and receiving processes.

The Intel 8086 was selected for the development project prototype because at the time it was the only single chip instruction processor available with a floating-point coprocessor (the Intel 8087). The system currently operates at a 5Mhz clock rate, limited by the 8087, although it is designed to run at 8 Mhz when faster 8087 chips become available. Processor element storage is a 128K-byte dynamic RAM which includes parity checking, but not error correction. Each node also includes 8 Kbytes of read-only storage for initialization,

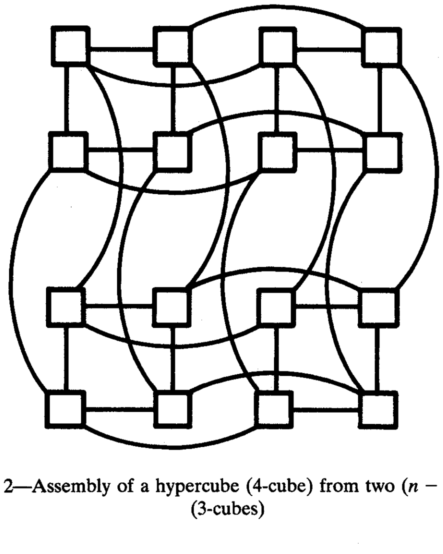


Figure 2—Assembly of a hypercube (4-cube) from two $(n - 1)$-cubes (3-cubes)

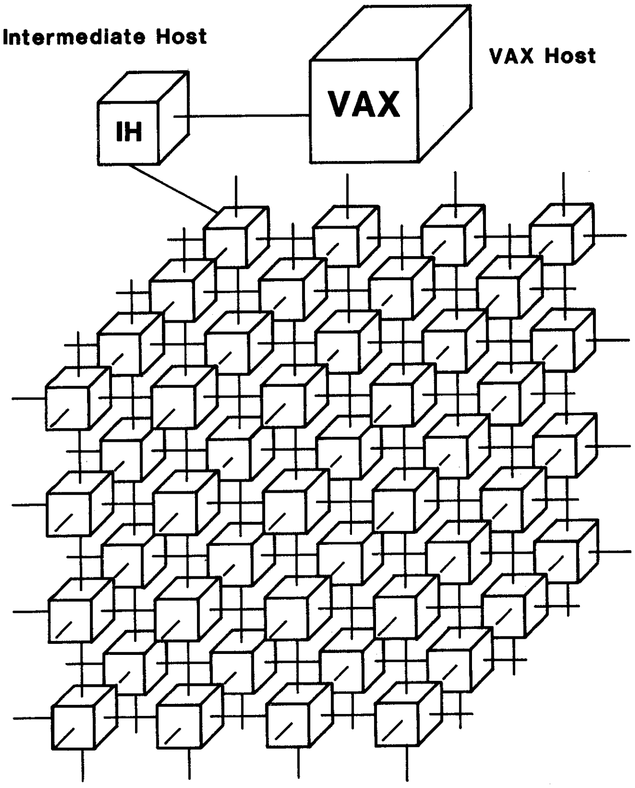


Figure 3—The Cosmic Cube six-dimensional hypercube architecture

bootstrap loading, dynamic RAM refresh, and diagnostic programs.

It should be noted that the current 64-node machine is isomorphic to a 4 × 4 × 4 cube with periodic faces, but is not restricted to this topology; for example, it can be configured as 4 × 4 × 2 × 2 or 8 × 8 meshes or a 64-processor ring depending on application requirements. The maximum communication distance between processors grows only logarithmically with the total number of processors.

Message Passing

A significant difference between the Cosmic Cube and many other MIMD parallel processor architectures is that this machine uses message passing rather than shared variables for communication between concurrent processes. The hardware includes no shared storage, no global addresses, and no switching network between processors and storage. Message passing is also handled by explicit communication and synchronization primitives invoked by the programmer. A computation is programmed as a collection of communicating processes which are simply instances of sequential programs that include actions of sending and receiving messages. A process is constrained to fit entirely within a single node. All processes execute concurrently, either in different modes or by being scheduled within a single node.[20]

When the Cosmic Cube is run with its standard operating system, each process has a unique identification number (ID) that serves as an address for sending it messages. This ID is a 16-bit integer that includes the node number and the process number within the node with an accompanying restriction that once a process is instantiated, it may not migrate to another node. Sending or receiving a message does not suspend execution; however, a process that is in a situation where no progress can be made until a message area is filled or emptied may elect to sleep. Synchronization between message sending and receiving may be quite loose, due to queueing occurring in the communication subsystem.

Language

The Cosmic Cube has no special programming notation. Process code is written in ordinary sequential programming languages (eg., FORTRAN, PASCAL, C, etc.) and extended with statements or external procedures to control actions such as the sending and receiving of messages. These programs are compiled on other computers, and are loaded into and relocated within the MIMD nodes as binary code, data, and stack segments.[20]

The programmer is required to formulate and express a computation explicitly in terms of a collection of communicating processes, and to control the mapping of processes to nodes well enough to achieve satisfactory concurrency and load balancing. The positive aspects of this manual technique include expressive freedom and control over the efficiency of the computation; the negative aspects include the fact that the programmer has a lot to keep track of, and that he may create programs with subtle bugs.[20] Research in concurrent programming and parallel processing at MCC has led to the position that concurrent programming languages based on extensions of traditional procedural languages do not have sufficient expressive power to support the application of a scalable parallel processing technology. While there are certainly rewards to be gained from the application of extended FORTRAN or concurrent C to problems characterized by a high degree of physical or logical partionability, the longer range prospects for concurrent programming lie with languages that have greater expressiveness, encourage natural concurrent programming styles and support the automatic extraction of parallelism inherent in the application.

Software

Programs for the Cosmic Cube, developed on a conventional sequential computer, include writing, compiling, simulating, instrumenting, and debugging. The programs are then downloaded through a connection that is managed by an intermediate host.

Node software consists of a startup packet which specifies the size of the cube to be initialized, and may specify that the built-in RAM tests be run concurrently in the nodes. A part of the initialization procedure involves each of the identical nodes discovering (by sending messages) its position in whatever size cube was specified in the startup packet sent from the intermediate host. This initialization involves messages that also check the function of all of the communication channels to be used. Program loading following initialization typically loads either a more sophisticated loader or the operating system.

The *Crystalline* applications environment is characterized by programs, written in C, in which there is a single process per node, and in which messages are sent by direct I/O operations to a specified channel. The standard operating system supports a multiple process environment with a small operating system kernel running in each node. Message routing and queueing is accomplished by the inner kernel, together with scheduling. The outer kernel provides process spawning, storage management, I/O communication, and monitoring and debugging facilities through special hooks to the inner kernel and by communication with the intermediate host.

It should be noted that in this environment, the definition and execution of the multiple process formulation of the problem is independent of its mapping onto physical nodes. Mapping influences only efficiency. It should also be mentioned that an essential feature of the programming environment is a simulator which allows researchers to run programs on the host with each node of the concurrent processor simulated by an individual process.

SUMMARY OF THE PRINCIPAL RESEARCH THRUSTS AND ISSUES

The 64-node Cosmic Cube has proven so useful for scientific computations that several evolutionary developments of this design are being constructed including two 16-node, a 32-node, and a 128-node machine. In addition, a 1024-node system using Mosaic chips is under development.

CalTech found that all of the scientific and engineering computationally-intensive problems which were examined could be efficiently implemented on concurrent VLSI processors in a hypercube architecture. The C^3P research staff stated that concurrent processors are quite easy to use, are not specialized devices, and can address the majority of computationally intensive problems. The basic difficulty is to decompose the problem into many parts or subdomains in such a way that each processor node is responsible for a single region. The hypercube node connectivity scheme is attractive because it includes the ring and many different mesh topologies as subsets, and it is an ideal topology for the fast Fourier transform.[13]

The two major problems faced when considering speedup are communication and load balancing. By having the luxury of defining the machine topology, the Cosmic Cube user is able to define a special configuration for any given algorithm which provides the "minimum" internode communication paths. For many algorithms, load balancing is achieved naturally, but in the non-homogeneous case, load balancing is much more difficult to achieve. At present, C^3P researchers perform the load balancing tasks manually; however, dynamic load balancing is not viewed as an insurmountable difficulty. In fact, at present C^3P is also developing a novel operating system which will run within each node of the concurrent processor. The Concurrent Computation Project has completed conceptual design of their next generation machine, the Mark III. It will use the new (64-bit) WEITEK floating-point chip, and probably Motorola 68020 and 68881 microprocessors. Each node will contain 1 to 4MB of memory and 10 internal channels so that systems up to 1024 nodes (10-dimensional hypercube) can be configured.

RESEARCH METHODOLOGY

The important feature of the hypercube architecture is that it is able to include all topologies that seem useful in the class of problems studied. Table I gives a list of the research areas being actively pursued. C^3P requires that the demonstration

TABLE I—Current areas of active research

Field	Scientist	Algorithm
Astrophysics[2,3,5,11,19]		
Evolution of universe	Quinn, Solomon	Fast Fourier transform/Long range force
Galactic dynamics	Otto, Warner	Fast Fourier transform
Black hole structure	Meier (JPL)	Finite difference and finite element
Chemical reactions	Kuppermann	Matrix inversion
Computer science circuit simulation	Mattison, Seitz	Irregular sparse matrix
Fluid dynamics	Keller, Saffman	Finite element methods/Finite difference
Geophysics		
Exploration	Clayton	Finite difference
Plate tectonics	Hager	Finite element
Granular motion	Haff, Werner	Semi-regular short range particle interaction
High energy physics[2-4]	Otto, Stolorz	Regular Monte Carlo
Image processing[18]	Solomon, Wall (JPL)	Filtering
Molecular dynamics	Shapiro	Long range force
Statistical physics[4-6,17]		
2D Melting	Fox, Johnson	Semi-regular medium range Monte Carlo
Coulomb gas	Solomon	Irregular long range force Monte Carlo
Structural mechanics[8,10]	Salama (JPL)	Finite element

problems to be tackled be at the forefront of research and that the entire problem (rather than a part or parts of it) be implemented on a concurrent processor.

CONCLUSION

The CalTech Concurrent Computation Project has developed a binary n-cube interconnection architecture based on concurrent VLSI microprocessor/coprocessor technology. A variety of applications programmed and solved demonstrate the high performance at low cost that advocates of parallel processing have expected. Several algorithms based on a new parallel model of computation were programmed in extended FORTRAN for execution on the hypercube under the supervision of a novel operating system. A programming technique has been developed to aid the manual extraction and representation of parallelism in the application. The hypercube is an important architecture for the development of a scalable parallel processing technology based on the interconnection of 16- or 32-bit microprocessor chip sets. The scalability features of the binary n-cube topology are attractive as is the maximal message path length which grows as $\log n$. While the current implementations are message-passing systems with no provision for global memory, such a feature can be accommodated by an $(n + 1)$-th port on each node or processor board for access to a common memory. The richness of the hypercube architecture and its scalability are strong factors in its application to functional language programming styles.[21] While the interconnection architecture is rich and eminently scalable, it is not always obvious how parallel applications of arbitrary topology can be mapped onto it. Cohn and Sullivan have shown that even worst-case mappings can reduce potential speedup by a factor of only two.[22] In contrast, the hypercube architecture offers the promise of significant performance advantages over sequential processors of the same cost.[23]

CalTech has set up a consortium of industrial partners to aid the transfer of this parallel programming technique developed in the course of the Concurrent Computation Project. The architecture and its special operating system have been licensed to at least one manufacturer. Presumably, organizations which purchase the commercial version will join the CalTech consortium to gain the advantages of being in a users group.

REFERENCES

1. Seitz, C. L. "Concurrent VLSI Architectures." *IEEE Transactions on Computers,* C–33 (1984), 12, pp. 1247–1265.
2. Brooks, E., G. Fox, M. Johnson, P. Stolorz, W. Athas, E. DeBendictis, R. Faucette, C. Seitz, and J. Stack. "Pure Gauge SU(3) Lattice Theory on an Array of Computers." *CALT–68–1112,* California Institute of Technology, Pasadena, March 1984.
3. Otto, S. W. "Lattice Gauge Theories on a Hypercube Computer." *CALT–68–1113,* California Institute of Technology, Pasadena, April 1984.
4. Fox, G. C. "Decomposition of Scientific Problems for Concurrent Processors." *CALT–68–986,* California Institute of Technology, Pasadena, March 1983.
5. Fox, G. C. "Concurrent Processing for Scientific Calculations." *Proceedings of COMPCON, 84, Twenty-eighth IEEE Computer Society International Conference:* Computer Society Press, New York 1984.
6. Fox, G. C. "Matrix Operations on the Homogeneous Machine." *CALT–68–939,* California Institute of Technology, Pasadena, August 1982.
7. Rapp, D. "Concurrent Algorithm for the Exchange Method of Inverting a Matrix." *JPL D–1538.* April 1984.
8. Salama, M., S. Utku, and R. Melosh. "Algorithms for Concurrent Processing." *CCP Report Hm–43,* California Institute of Technology, Pasadena, November 1983.
9. Fox, G. C., and S. W. Otto. "Algorithms for Concurrent Processors." *Physics Today.* May 1984.
10. Salama, M., and J. McGregor. "Parallel Solution of Finite Element Equations." *CCP Report Hm–18c,* California Institute of Technology, Pasadena, February 1983.
11. Salmon, J. "An Astrophysical N-body Simulation on the Hypercube." *CCP Report Hm–78,* California Institute of Technology, Pasadena, September 1984.
12. Hockney, R. W., and C. R. Jesshope. *Parallel Computers.* Bristol, U.K.: Adam Hilger, 1981.
13. Pease, M. C. "The Indirect Binary N-cube Microprocessor Array." *IEEE Transactions on Computers,* C–26–5 (1977), pp. 458–473.
14. Millard, W. "Hyperdimensional Microprocessor Collection Seen Functioning as a Mainframe." *Digital Design,* November 1975, p. 20
15. Lang, C. R. "The Extension of Object-oriented Languages to an Homogenous, Concurrent Architecture." *CalTech 5014:TR:82,* California Institute of Technology, Pasadena, 1982.
16. Seitz, C. L. "The Cosmic Cube." *Communications of the ACM,* 28-1 (1985), pp. 22–33.
17. Fucito, F., and S. Solomon. "Monte Carlo Parallel Algorithm for Long Range Interactions." *CALT–68–1134,* California Institute of Technology, Pasadena, June 1984.
18. Fucito, F., and S. Solomon. "On the Relation Between the Coulomb Gas and the Lattice XY Model." *CALT–68–1114,* California Institute of Technology, Pasadena, April 1984.
19. Meier, D. "Two-dimensional, One-fluid Hydrodynamics: An Astophysical Test Problem for the Nearest Neighbor Concurrent Processor." *CALT–68–XXXX,* California Institute of Technology, Pasadena, July 1984.
20. Lyzenga, G. "The Nearest Neighbor Concurrent Processor: A User's Tutorial Guide." *CCP Report Hm–68,* California Institute of Technology, Pasadena, August 1984.
21. Vegdahl, S. R. "A Survey of Proposed Architectures for the Execution of Functional Languages." *IEEE Transactions on Computers,* C–33–12 (1984), pp. 1050–1071.
22. Cohn, L. A., H. Sullivan, and T. M. Bashkow. "Uniform Bounds on Efficient Parallel Execution Time and Speedup." Microelectronics and Computer Technology Corporation, Austin, Texas, December 1984.
23. D. Abmayr, Ed. "A Catalog of Parallel Processing Programs and Products." Version 1.0, Microelectronics and Computer Technology Corporation, Austin, Texas, February, 1985.

An architecture for doing concurrent systems research

by VASON P. SRINI
University of California
Berkeley, California

ABSTRACT

An architecture for experimenting with different algorithms in numeric/symbolic computing, concurrent programming paradigms, and interconnection networks is proposed. The architecture comprises a large collection of processors organized in groups called *levels*. Each level has interconnection networks to communicate with memory units in the level and also a global memory called *system memory*. Each memory unit in the architecture has two parts, data memory and synchronizing/status memory. The latter is identical to the former except for the word-size. Separating synchronizing/status information from data reduces memory contention and improves the cycle time to memory for reading or writing.

Two proposals for processing elements are also outlined. One of them is a custom design and the other is a commercial processor with the addition of functional units capable of doing floating-point and symbolic calculations. The architecture will have facilities to support experimentation with three classes of concurrent computation models, a task level dataflow, demand driven, and cooperating sequential processing; programming languages such as Prolog with AND parallelism and built-ins for scientific computation will be supported by the architecture along with other languages. An initial design of the architecture using crossbars as interconnection networks is also outlined.

INTRODUCTION

Concurrency is a key area in computer science that can use VLSI technology to achieve improved performance. Concurrency can be specified in the model of computation and algorithms, and supported by the computer architecture. Although concurrency has been studied in detail, there are very few facilities for experimenting with concurrent architectures, interconnection structures, and software. Existing systems such as Cm* at Carnegie-Mellon University and TRAC at the Univ. of Texas in Austin are difficult to program due to program and data distribution problems. For example, the Cm* architecture supports fast, slow, and very slow memory. Distributing programs and data so that time to obtain data from memory can be reduced is difficult. In the TRAC architecture, the memory cycle time is slow because of the switch between memory and processors. Furthermore, both lack the potential to evaluate the execution overheads of different models of computation. This paper describes a computer systems research facility that can support concurrent systems research in scientific computing aided by symbolic/logic processing. The objectives of the proposed computer system are:

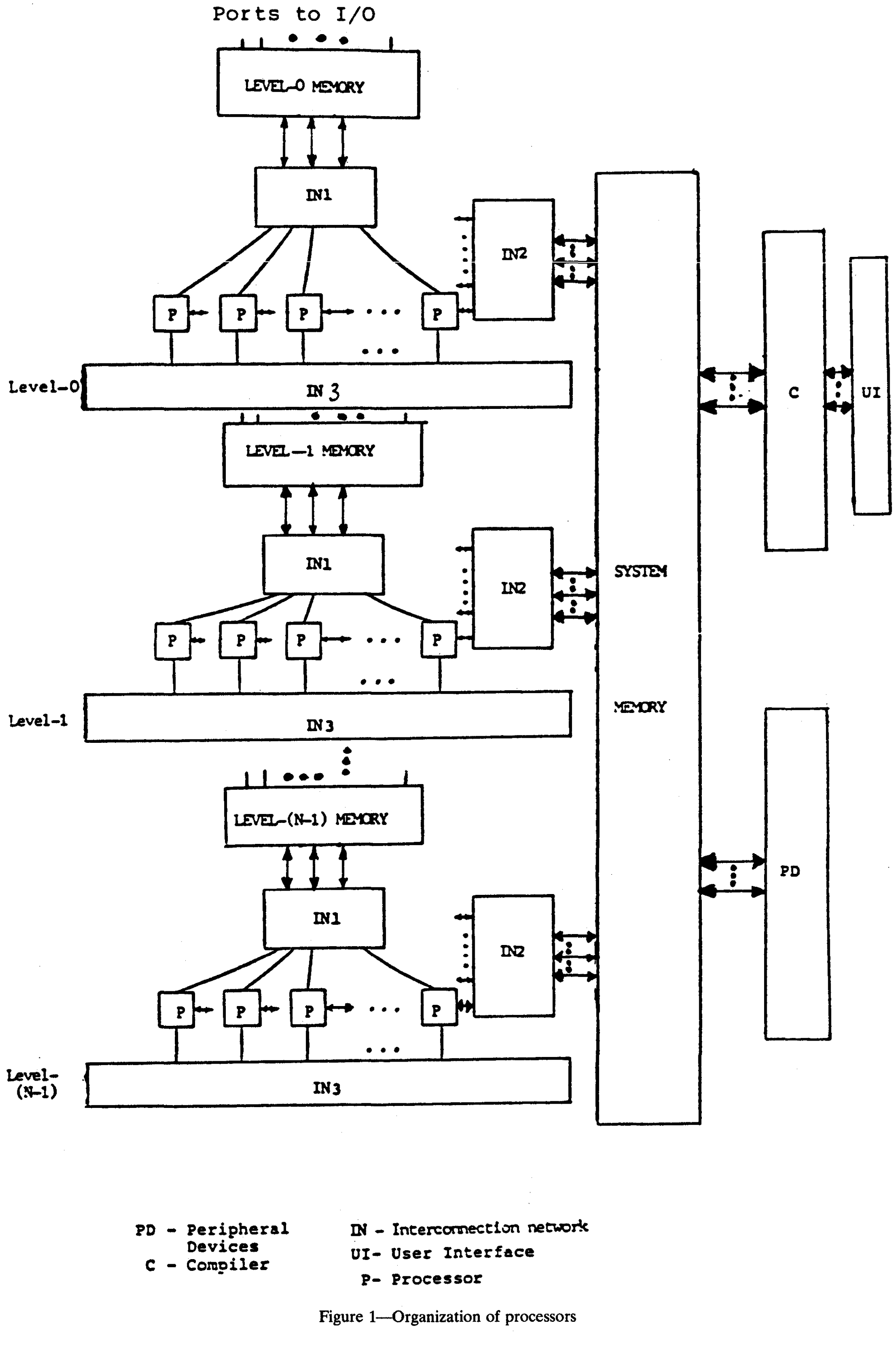
1. To develop a system using several hundred processors to support the execution of numeric and symbolic algorithms coded in functional, logic, and control flow programming styles
2. To evaluate interconnection structures such as integrated bus-star, integrated bus-ring, integrated ring-star, global bus, crossbar, omega network, and banyan
3. To develop strategies for distributing programs and data in a semi-automatic way
4. To determine the overhead in executing programs on concurrent architectures and design tradeoffs related to this overhead
5. To develop concurrent programs to solve problems in areas such as fluid dynamics, aerodynamics, weather prediction, computational physics (e.g., simulating the superconducting super collider (SSC)), and computational chemistry, and to develop CAD tools for designing concurrent architectures

An outline of the systems architecture and the computation models that will be supported by it are discussed next. Two key features of the system are the separation of synchronizing/status memory from data memory, and the use of clusters of processors to support spatial and temporal locality in programs.

SYSTEMS ARCHITECTURE

There are three major models of computation that have the characteristics of concurrency and asynchrony; they are dataflow,[1–3] demand driven,[4, 5] and Hoare's cooperating sequential processes.[6] The dataflow and demand driven models also have functionality; that is, the execution of an activity represented as a node in the dataflow graph consumes inputs and produces outputs. There are no side-effects in the execution. Hence, program maintenance and distribution can be done without excessive overhead. An organization of processors that is expandable in a modular way and capable of supporting the three models of computation is shown in Figure 1. The synchronizing/status memory parts of memory units are not shown in the figure. The shadow of the drawing (obtained by duplicating the entire figure) can be thought of as the synchronizing/status part. Implementation is planned for the support mechanisms needed for the execution of programs on the processors in Figure 1 as the kernel functions of an operating system. The support functions accomplish task allocation, token communication, and node reassignment when processors fail. There are several reasons for using software instead of hardware to implement the support functions. A key reason is that experiments can be conducted using different policies for the support mechanisms. The kernel functions will be executed on a subset of (three or four) processors in each level. The kernel functions needed to execute programs using the dataflow model of computation are shown in Figure 2. By changing the semantics of the kernel functions, different strategies for task allocation, token construction, and token communication can be studied. The overhead in executing tasks, and the design tradeoffs related to the overhead can also be studied. By changing the structure of the kernel, the other models of computation can be supported. A detailed description of the organization of processors in Figure 1 and the kernel functions in Figure 2 are in the papers by Srini.[7–9] The organization of processors into levels allows high speed interconnection networks such as crossbars to be used for IN1-IN4 in Figure 1. It is believed that 32 × 64 crossbar chips with 400 pins can be fabricated and packaged using state-of-the-art technology. The levels of processors also support locality and simultaneity of execution in concurrent programs. For example, blocks of code operating on data structures such as arrays and lists contain spatial locality. Unfolding of loops and having many instances of the loop in concurrent execution provide temporal locality.

There are several programming paradigms that can be realized using the above system. We will focus on functional pro-



PD - Peripheral Devices
C - Compiler
IN - Interconnection network
UI- User Interface
P- Processor

Figure 1—Organization of processors

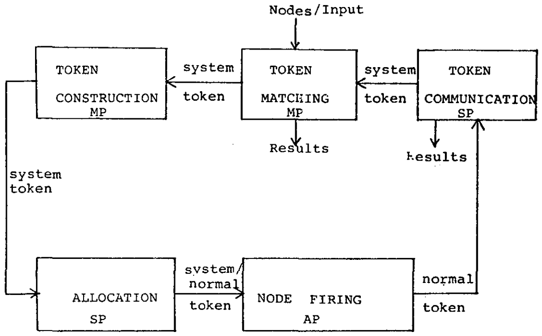


Figure 2—Program execution on the architecture. *Allocation* assigns a process to a node and processor. *Token construction* constructs a token, using opcode and operands. *Node firing* executes the operation assigned to a node. *Token communication* determines the destination for results and forwards the result to the destination.

gramming,[10, 11] logic programming,[12–14] and control flow programming[15] because they have the potential to express concurrency. The mapping from the programming paradigms to the three models of computation are explained by Vegdahl[11] and Treleaven.[15] The purpose of the proposed architecture is to support the experimentation of multiple computation models with different programming paradigms. For example, functional programs will be executed using dataflow, demand-driven, and control flow architectures, and the execution times will be analyzed to determine the overheads and bottlenecks due to the computation model. Similarly, logic programs with AND, OR, and stream parallelism can be executed using dataflow and control flow architectures, and the execution times can be analyzed to determine the overheads. As a result of the experiments, high performance architectures suitable for the three programming paradigms will emerge.

ORGANIZATION OF PROCESSORS

There are several possible ways to implement the logical organization of processors shown in Figure 1. One implementation was suggested in the dataflow processor proposal.[8, 9] Another implementation is currently being investigated by researchers headed by Despain at the University of California at Berkeley.[16] The key ideas are the use of VLSI chips for the interconnection network to reduce delay and wiring complexity, the use of processors with specialized functional units to support the computation models, and the use of operating system functions to perform the flow control. Memory is separated into data part and synchronizing/status part to reduce memory interference and speed up synchronization. The word length for the data part is 32 bits, and for each word there is a byte in the synchronizing/status part. The memory units in the proposed system are byte addressed.

An initial design for the architecture in Figure 1 is outlined below. The numbers chosen were based upon the feasibility of constructing the architecture and its usefulness for experimentation. The architecture uses crossbars instead of buses and rings to get fast access to all the memory units so that communication costs do not clutter other overhead issues. With a crossbar, it is possible to get the interconnection delay to be around 20% of the memory cycle time. The memory contention problem in the crossbar will be handled in a distributed way at the processors. A processor experiencing memory contention will proceed with other activities and reattempt at a later time. It is believed that a crossbar with 50 nanoseconds delay can be fabricated by a corporation such as NCR using an advanced CMOS process, and that 256K memory chips with a cycle time of 250 nanoseconds will be cheap in 1985. The number of processors and memory units in a level are dictated by the size of the crossbar. The initial design of an 8 × 8 crossbar chip leads us to believe that a 16 × 32 crossbar chip can be designed and fabricated in 1985. The size of the memory units are determined by the memory requirements of application programs in the scientific computations. All the data paths are assumed to be 32 bits wide. This is not a restriction for doing some of the scientific calculations involving 64-bit data items. The data paths can be readily expanded by adding 32 single-bit VLSI crossbar chips to each interconnection network in the architecture.

1. Each level memory is a set of 32 memory modules, each with a 4 Mbyte capacity and two ports. The second port can be used for moving data to and from I/O devices. Corresponding to each of the above modules, there is a module in the synchronizing/status memory with a capacity of 1 Mbyte (one byte for each 32-bit word in the data memory). The 4 Mbytes of data memory can fit on the single hex board used by DEC if 256K memory chips are used. These boards are used in the VAX machines and are commercially available.
2. Each level has 16 processors; most are general purpose and some are special purpose processors.
3. The system memory is a shared one, having 12 ports (eight ports for eight levels of processors and four for I/O processors and front-end) for each module. It contains up to 64 modules, each with a capacity of 16 Mbytes. Corresponding to each of these memory modules is a memory module in the synchronizing/status memory with a capacity of 4 Mbytes and 12 ports. The 16 Mbytes of data memory can fit on a single DEC hex board if 1M memory chips are used. These boards are expected to be commercially available in 1986. The 12 ports can connect up to eight levels of processors, three input and output processors (see PD in Figure 1), and one interface to the front-end running the compiler. The system memory has four 16 × 64 crossbars to connect the I/O and front-end processors with the data part of the system memory. An additional set of four 16 × 64 crossbars connect the above processors to the synchronizing/status memory part of the system memory. This organization allows high speed communication of data to I/O devices.
4. The interconnection network IN1 contains a 16 × 32 crossbar with 32 bits wide data and address paths to the data part of level memory. It also contains a 16 × 32 crossbar with an 8-bit-wide data path and a 32-bit-wide address path to the synchronizing/status part of level

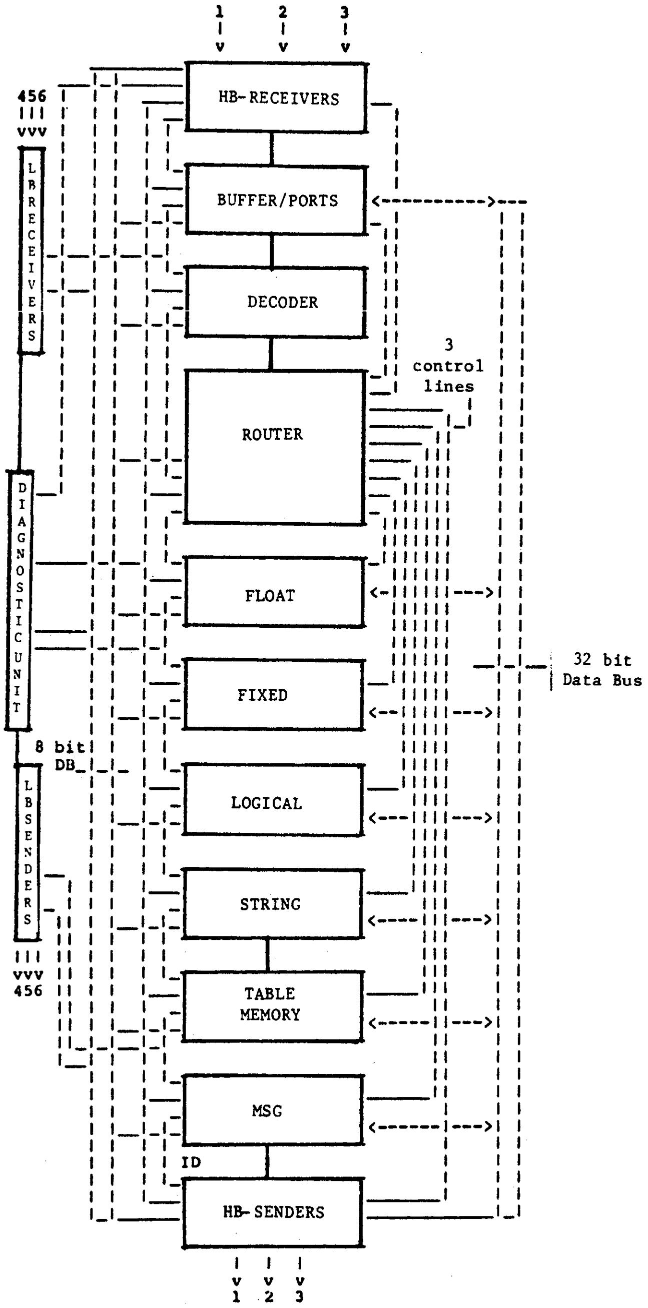
memory. The crossbar will be built using single-bit-slice chips. Each chip contains 182 pins. This pin count is based on the Macpitts design of the chip. Macpitts is a silicon compiler developed by the Lincoln Laboratory and modified by researchers at the University of California at Berkeley to generate layout for special purpose applications. An interconnection network similar to IN1 can be used on the second ports of the level memory modules to connect the I/O controllers and devices.

5. The interconnection network IN2 is a 16 × 64 crossbar with 32-bits-wide data and address paths to the data part of system memory. It also contains a 16 × 64 crossbar with an 8-bit-wide data path and 32-bit-wide address path to the synchronizing/status part of system memory. This crossbar will also use single-bit-slice chips. Each chip needs 263 pins. This pin count is also based on the Macpitts design of the chip.
6. The interconnection network IN3 contains a 16 × 32 crossbar with 32-bit-wide data and address paths and a collection of 16 console debuggers, one for each processor in a level. The 32 output ports of the crossbar will be used in the following way. Sixteen ports will be connected to the processors and 16 to console debuggers. Thus, the crossbar in IN3 allows processors to communicate within a level. Using the console debugger, a processor can communicate with the external environment. There is another 16 × 32 crossbar to connect the synchronizing/status ports of the processors (see LB paths in Figure 3), and console debuggers.

The efficiency of communication facilities is an important area in concurrent systems research. For example, there are three interconnection networks in Figure 1. The performance of the computer system shown in Figure 1 depends on the interconnection networks, processor architecture, and the kernel functions. By using different interconnection networks for IN1-IN3, and separating data from synchronization/status information, the efficiency of communication facilities can be determined. It is believed that the above implementation using crossbar will provide insight into predicting the efficiency of other communication facilities such as the Omega network. The architecture will also allow us to study the performance gain obtained by separating the data and synchronization/status information and keeping them in separate memory units.

PROCESSOR DESIGN

Efficient execution of programs containing numeric and symbolic computation rquires processors with an instruction set containing logical, fixed-point, floating-point, string manipulation, and sorting operations. Other features required are data cache, mechanisms for communicating with memory, and fault diagnosis. There are at least two ways to develop processors with the above characteristics; one is to custom design a processor, the second is to use a commercially available processor and interface special purpose units to it.



DB = Diagnostic bus
ID = Interrupt line daisy chain
1 = HB path to processors
2 = HB path to level memory
3 = HB path to system memory
4 = LB path to processors
5 = LB path to level memory
6 = LB path to system memory

Figure 3—Units of the EDFG processor

Custom Design

The block diagram of a custom designed processor that satisfies the above requirements and supports the dataflow model is shown in Figure 3. The processor is capable of executing dataflow graphs generated from functional programming languages and logic programming languages. The HB paths in the processor communicate data. The LB paths communicate synchronizing/status information. By employing user microprogrammable functional units in the processor, it will be possible to experiment with different instruction sets for the processor. Some of the key features of the processor and the units are explained next. For further details, see my previous description of this processor.[9]

The processor receives from a memory unit an enabled node for firing as a sequence of packets forming a variable length message (token). Data may be contained in the token, or it may arrive separately in another token. A node's operation can contain a maximum of five instructions. The processor performs node firing and sends the results to other processors as specified by the kernel functions doing the allocation of nodes to processors. The various parts of the message-based processor shown in Figure 3 are now described. All units of the processor will eventually have built-in diagnostics and perform self-diagnosis. Diagnostic information will be communicated to the diagnostic unit.

HB-receivers (LB-receivers): This unit receives tokens containing data (synchronizing/status information) from the interconnection network, and stores it in the buffer/ports unit.

HB-senders (LB-senders): This unit sends tokens containing data (synchronizing/status information) to the interconnection network. In the case of crossbar for interconnection networks, the memory contention handling and retry logic are contained in this unit.

Decoder: This unit decodes instructions received in a token, and sends control to the router for supplying operations and data to the various functionai units.

Router: The operations and data supplied by the decoder are routed to the functional units using a receive/ack/last protocol. Three control lines are used for the protocol to each of the seven functional units.

Message (MSG) unit: This unit receives and sends tokens to memory on the HB and LB paths. Requests for manipulating structures and tables in memory are also handled by this unit.

Buffer/Ports: The processor contains two 512-bit buffers for storing tokens; one buffer will contain the token that is acted upon by the processor, the second can be used for storing tokens arriving from memory as a result of read requests by the processor, and for storing the outputs of functional units.

Table memory: This is an associative/random-access memory with 64 entries, each 64 bits wide. For each entry there is a 6-bit index field. Given a 64-bit value, the table memory supplies the contents of the index field corresponding to the entry that matches the 64-bit value. Multiple matches are indicated by the table memory. In the event of a multiple match, the entry with the lowest index value is selected. It is also possible to read or write a 64-bit word using the 6-bit index.

FLOAT (Floating-point) unit: This 64-bit floating-point unit is capable of performing floating-point operations using IEEE Standard 754. It has two input ports and an output port. When data is available in both input ports, the operation assigned to the unit is performed and the result stored in the output port. Thus, the unit is data driven.

STRING unit: This 64-bit string unit with two input ports and a third port that can act as an input or output port is also data driven. Some of the operations performed by the unit need only a single operand while others need two or three. The unit uses table memory to store strings and search or sort as a part of some of the operations. Several useful string operations are supported as instructions. In addition, activity name generation and manipulation operations are also supported.

FIXED unit: This data driven 32-bit unit with two input ports and one output port performs integer operations. Usual operations such as add, subtract, multiply, and divide are supported as instructions. In addition, adding a list of numbers and multiplying a list of numbers operations are also supported.

LOGICAL unit: This data driven 32-bit unit with two input ports and a third port that acts as an input or output port performs some of the common logic, shift, comparison, and control operations of dataflow, supported as instructions.

Diagnostic unit: This unit communicates with all the parts of a processor to gather diagnostic information. It decides whether the processor is healthy or faulty on the basis of the gathered information. Urgent tokens containing diagnostic information from memory is received by this unit. Diagnostic information is sent to memory and to other processors in the form of urgent tokens. This unit also reroutes tokens stored in the buffer/ports unit. Note that the diagnostic unit is not the hard core of a processor, since the functional units do their own diagnostics.

A bus architecture is used in the processor for communication between the table memory, floating-point, fixed-point, logical, and string units. The functional units are user microprogrammable so that new instructions can be added and microdiagnostics can be used. The processor has been designed using AMD 2900 bit-slice chips to get estimates on the execution times for different instructions. A VLSI version of the processor will be custom designed employing Weitek's 1064/65/66 floating-point chips.

NCR/32—PLM Design

The block diagram of a processor using a commercially available microprocessor is shown in Figure 4. It contains an

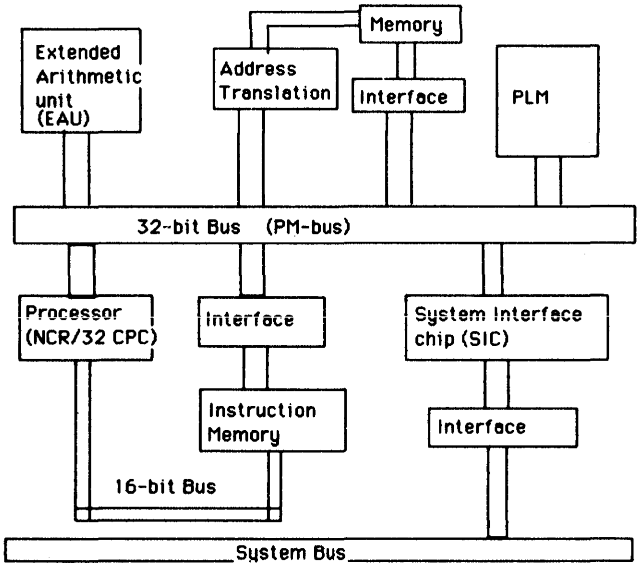


Figure 4—Processor using NCR/32 and PLM

NCR/32 central processor chip (CPC),[17] a user microprogrammable unit to do the usual functions of a processor, a floating point unit, and a Prolog functional unit. The logic-programming language Prolog is selected as an initial language for this design because it has the potential for execution on a dataflow and a control flow architecture. It is also the language used by the high-performance systems research group at UC Berkeley (headed by Despain) and the Japanese Fifth Generation Project. Prolog has several features that are found in functional programming languages and block structured languages such as Pascal and Modula 2. It has been conjectured by Despain and other researchers that the class of functional programming languages is a subset of the class of logic programming languages. It is believed that an understanding of the execution environment for functional and control flow programming languages can be obtained by starting with Prolog. A Prolog program execution unit called PLM[18] is planned to support symbolic processing. An extended arithmetic unit (EAU) is included to perform integer multiply and divide, floating-point operations, and trigonometric functions. A version of this processor is under development at the Computer Science Division of the University of California at Berkeley.[18]

The PLM executes the instructions of an abstract machine developed by Warren.[19] The PLM has five classes of instructions to execute Prolog programs and to invoke the NCR/32 to do scientific computations. Some of the instructions perform unification of a goal's arguments with a clause-head's arguments, moving of a goal's arguments in registers to memory or other registers, and moving of the arguments of a goal in a clause's body from memory to registers. Other instructions create a new environment for a procedure call, save the context for backtracking, discard the current environment, backtrack, and branch instructions based on the data type in an argument register to select a clause in a procedure. The built-in functions of Prolog that are related to scientific computation are not performed by the PLM. Rather, an escape mechanism is provided in the PLM architecture to communicate the built-ins to the NCR/32. The built-in functions are computed using the NCR/32 and the EAU. A compiler written by Van Roy[20] generates code for the PLM from Prolog programs.

The NCR/32-based design is an inexpensive design created to aid in understanding some of the research issues in interconnecting processors to memory and designing software. To provide high performance in doing scientific computations, the Convex C-1 processor[21,22] can be used. It is a 64-bit processor with a virtual memory space of 2 Gbytes per process and a maximum of 2 Gbytes of physical memory. One half of the physical memory is reserved for I/O. The CPU and I/O communicate with memory using two 80 Mbyte buses (64-bit data path) and a memory controller. It appears that an interconnection network can be placed between the CPU and memory as well as between the I/O and memory without degrading memory accesses. The synchronizing/status memory can be added by tapping the addresses that appear on the two buses and sending them to the synchronizing/status memory, or by using the I/O space. Because Unix 4.2 bsd is running on the Convex C-1 processor, most of the software developed for the NCR/32-based design can be used.

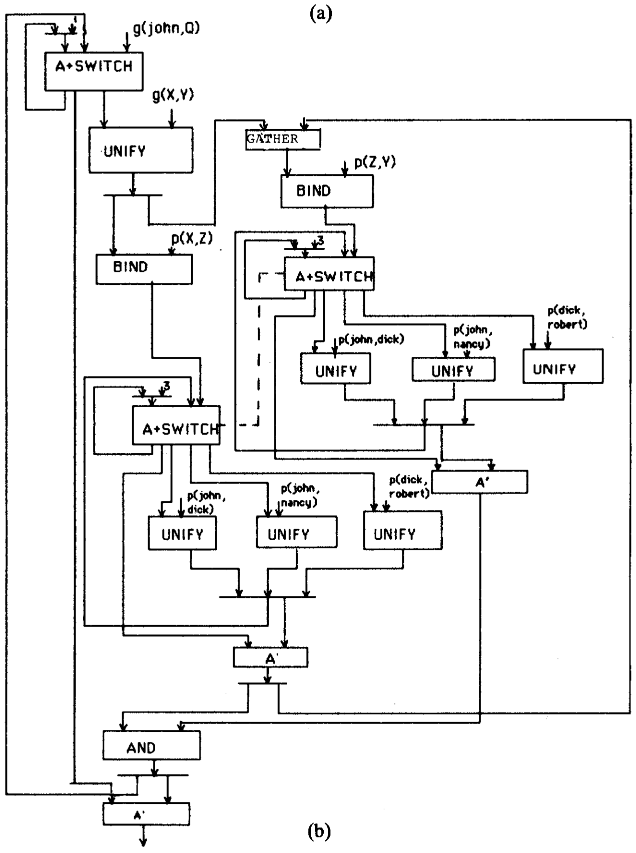
Support For AND Parallelism

Concurrency in Prolog programs can be exploited by using AND, OR, and stream parallelism.[23] Exploiting OR and stream parallelism in Prolog programs needs hardware support because supporting them in software requires excessive overhead. It is also difficult to control OR parallelism. Initially, AND parallelism in Prolog programs will be considered. At a later stage, stream parallelism will be exploited. The goals in the body of a clause with AND parallelism will be concurrently executed on a set of processors in a level.

Compiling Prolog programs with AND parallelism to the architecture presents some additional problems; the first is detecting the AND parallelism at compile time, the second is generating code for multiple processors that would keep track of the environment and the choice points without creating excessive memory interference. Although dynamic analysis of Prolog programs to detect AND parallelism has been reported by Conery,[23] it is an expensive approach and complicates the code generation for multiple processors. Recently, Chang[24,25] has devised a scheme to detect AND parallelism at compile time by annotating the programs at the top level. The above development has opened new possibilities in code generation. Figure 5 shows a sample Prolog program and the dataflow graph that can be generated for it. The data dependencies between the goals in a body is shown by directed arcs. These arcs resulted from the compile time analysis of the Prolog program. The dashed arcs in the graph represent communication paths for synchronization/status information. These arcs can also be supplied by the compiler but they require extra analysis. The nodes in the dataflow graph can be assigned to processors in a single level or multiple levels for node firing. By using a tagging scheme and special operators

```
g(X,Y):- p(X,Z), p(Z,Y).        QUERY:

p(john, dick).                   :- g(john,Q)?

p (john,nancy).

p(dick,robert).
```

(a)



(b)

Figure 5—(a), Prolog program; (b), dataflow graph for a Prolog program

such as "A + SWITCH" to save an old context, create a new context, and branch, recursive clauses and backtracking can be handled. Unification (a process of finding the most general common instance of two terms) is carried out by using nodes with the operation "UNIFY". Creating an environment for evaluating a literal (sub-goal) is done by using a node with the operation "BIND". Microroutines have to be developed for the above operations and others that support AND parallelism.

Application Areas

Programs from three application areas will be executed on the architecture to evaluate the performance of the system. The areas are scientific, knowledge-based expert systems, and CAD tools (e.g., silicon compilers, circuit simulators, routers, and placement programs for cells). The programs will be written in extended Prolog and other languages.

CROSSBAR DESIGN

One of the components that play a key role in the performance of the architecture in Figure 1 is the crossbar. Although crossbars have been used in computer systems for more than two decades, very few high-performance off-the-shelf chips are available. Some VLSI designs based on an incremental design have been proposed by Franklin,[26,27] and other designs are internal to corporations making signal processing sytems. In this paper, a high-performance fixed-delay (assuming conflict-free references) design for 8 × 16, 16 × 32, 16 × 64, and 32 × 64 crossbars is proposed. The design employs a single-bit-slice approach with a maximum of six gate delays in the switch part (not including delays in the pads and pad-drivers) and a maximum of 400 pins.

The design and the functioning of the crossbar is explained using the pin out of a single-bit-slice crossbar chip with 16 input ports and 32 output ports, as shown in Figure 6. The input ports are connected to processors, and the output ports can be connected to memory units or processors. For example, if the processors correspond to those shown in Figure 3, then the input ports will be connected to the HB-senders or LB-senders and the output ports will be connected to the HB-receivers or LB-receivers. Each input port contains five address pins, one address/data-in pin, and one data-out pin. Each output port contains a data-in pin and a data-out pin. Processors can communicate words of data and address by using multiple single-bit-slice chips. For example, by stacking 32 single-bit-slice chips, a 32-bit address/data can be sent to memory and a 32-bit data received from memory.

The design makes several simplifying assumptions. If two or more processors request the same memory unit, a simple policy is used to select one of them. One or more other processor(s) will receive signals indicating contention for memory. The processors are assumed to have the necessary logic

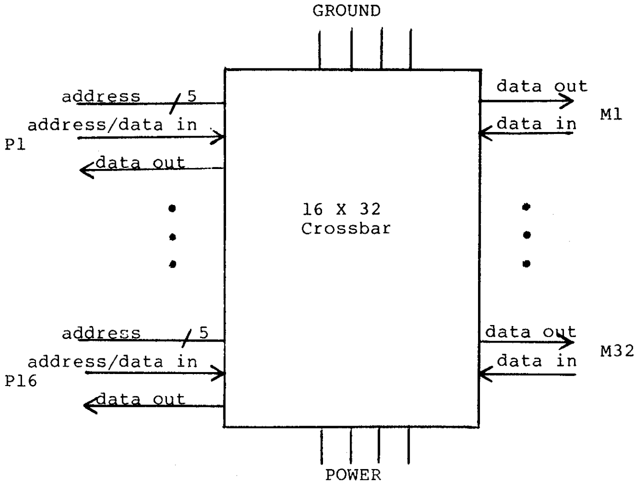


Figure 6—Pin out for a single-bit-slice crossbar chip

to handle contention and retry at a later time. Each processor is assumed to supply the memory unit or the processor number it wants to communicate, followed by an address and data in the case of a memory write, or just an address in the case of a memory read. The single-bit-slice chip in Figure 6 contains several multiplexers, selectors, and arbitration logic designed in static CMOS. The NAND gates used in the design have a fanin of 5 and the NOR gates have a fanin of 12.

A processor wishing to read/write from one of the 32 memory units supplies the address of the memory unit on the address pins. The processor then sends the address of the desired location in the memory unit using the address/data-in pins of the stack of chips. For write operations, the processor supplies data on the next processor cycle using the address/data-in pins. For read operations, the processor receives data from the selected memory unit on the data-out pins of the stack of chips. Each memory unit is connected to one output port (a data-in pin and a data out-pin) of a single-bit-slice chip. A memory unit receives an address followed by data from the data-out pins of the chips if a write to memory is to be performed. If an address is not followed by data, then the memory unit reads the contents of the desired location and supplies it on the data-in pins of the chips.

Note that the proposed design is different from the incremental design[26,27] in two respects. The flexibility of the incremental design is traded off for reduced gate delay. The complexity involved in contention handling is traded off to obtain a simple design. The price one has to pay for simplicity is degradation in performance when processors requests are skewed towards a subset of memory units. Assigning tasks to processors so that memory references will not be skewed towards some memory units is still an open problem. It appears that reassigning tasks to other processors at runtime can mitigate the problem.[8]

DISCUSSION

An architecture capable of supporting 128 processors is proposed. The architecture employs 16 × 32 crossbars and 16 × 64 crossbars for processor memory communication. VLSI chips for the crossbars seem feasible with static CMOS technology (2-micron feature size and two layers of metal) and pin grid array packaging. The use of crossbars in the system allows the cycle time to read (write) from either level or system memory to be the same. It is expected that having one cycle to memory should reduce the execution overhead and simplify program and data distribution. This feature of the proposed system should be contrasted with the fast, slow, and very slow memory accesses in Cm*[28,29] which makes program distribution a challenging task.

The proposed architecture can be used to map many of the multiprocessor architectures proposed by universities and industrial laboratories. For example, the Cedar multiprocessor system of the University of Illinois[30] can be mapped on to the architecture if each cluster is treated as a level of processors and the number of processors per cluster is limited to 16. The Omega networks will be replaced by the crossbars. The cluster control unit and the global control unit functions can be implemented using the synchronization/status memory units and one processor in each cluster. The mapping of Cedar architecture is important because thousands of benchmark programs analyzed by the Paraphrase compiler can be run on the architecture. The results will allow us to determine the performance penalty paid for multistage blocking networks such as Omega. The execution overheads for task level dataflow on control flow architecture and dataflow architecture can be compared. The static dataflow architecture of Dennis[31] can also be implemented on the architecture if the enabling count information of nodes are kept in the synchronizing/status memory units, and a few of the processors in each cluster execute functions corresponding to enabling condition detection.

Several issues must be resolved before constructing the proposed system. A key issue is the protocol to be used in reading or writing to data memory with the synchronizing/status memory in place. Another key issue is predicting the traffic between the processors and the memory for a class of algorithms and developing appropriate policies to be used in distributing programs and data. Another issue is the accessing of data structures (e.g., arrays, lists,and records). Efficient ways to read and update data structures must be devised. This is still a challenging problem in the dataflow and demand driven architectures. New languages have to be developed or existing ones augmented to exploit the multiple processors in the architecture. Another key issue is the design of the operating system, and predicting the overhead in the execution of programs on the processors.

Developing a detailed event-driven simulator for the architecture will be the first step of the project. The availability of VAX-11/784 with shared memory at LBL will allow us to get a head-start on the simulation activity. An analyzer for detecting AND parallelism in Prolog programs and a compiler for generating dataflow graphs are two other areas that will be carried out on the VAX-11/784.

ACKNOWLEDGMENTS

I am thankful to Professor Al Despain for his comments on the architecture and the crossbar design. The numerous discussions I had with Tep Dobry, Barry Fagin, Wayne Citrin, Jung Chang, Philip Bitar, and Robert Young have been invaluable in my understanding of Prolog and the PLM design. I am also thankful to NCR Corporation, Dayton, for funding parts of the Prolog machine research. I am thankful to Dr. Arie Shoshani for his comments on the paper and the discussions related to the architecture.

REFERENCES

1. Adams, D. A. "A Computation Model with Data Flow Sequencing." *TR-117.* Stanford University, CS Dept., School of Humanities and Sciences, Dec. 1968.
2. Dennis, J. B. and J. B. Fosseen. "Introduction to Data Flow Schemas." *Computation Structures Group Memo: 81.* MIT, Project MAC, 1973.
3. Arvind, and K. P. Gostelow. "The U-Interpreter." *Computer.* 15–2 (1982), pp. 42–50.
4. Abrams, P. S. "An APL Machine." Ph.D. dissertation, Stanford University, Feb. 1970.

5. Henderson, P., and J. H. Morris. "A Lazy Evaluator." *Proceedings of the ACM Symposium on Principles of Programming Languages.* 1976, pp. 95–103.
6. Hoare, C. A. R. "Communicating Sequential Processes." *Communications of the ACM.* 12–8 (1978), pp. 279–288.
7. Srini, V. P. "An Architecture for Extended Abstract Dataflow." *Proceedings of the 8th Annual Symposium on Computer Architecture.* Minneapolis, 1981, pp. 303–325.
8. Srini, V. P. "A Fault-tolerant Dataflow System." *Computer.* 18–3 (1985), pp. 54–68.
9. Srini, V. P. "A Message-Based Processor for a Dataflow System." *Proceedings of the 1984 International Workshop on High-level Computer Architecture.* College Park, Md.: University of Maryland, 1984.
10. Henderson, P. *Functional Programming Application and Implementation.* Englewood Cliffs: Prentice-Hall, Inc., 1980.
11. Vegdahl, S. R. "A Survey of Proposed Architectures for the Execution of Functional Languages." *IEEE Transactions on Computers.* C–23–12 (1984), pp. 1050–1071.
12. Kowalski, R. A. "Predicate Logic as Programming Language." *Proceedings of the IFIP Congress.* 1974.
13. Warren, D. H. D. "Logic Programming and Compiler Writing." *Software Practice and Experience.* 10, (1980), pp. 97–125.
14. Clocksin, W. F., and C. S. Mellish. *Programming in Prolog.* New York: Springer-Verlag, 1981.
15. Treleaven, P. C. "The New Generation of Computer Architecture." *Proceedings of the 10th Annual International Symposium on Computer Architecture.* Stockholm, 1983, pp. 402–409.
16. Dobry, T. P., J. H. Chang, A. M. Despain, and Y. N. Patt. "Extending a Prolog Maching for Parallel Execution," submitted to Logic Programming Conference, 1985.
17. NCR Corporation. *NCR/32 General Information.* NCR Microelectronics Division, Colorado Springs, Co., 1983.
18. Dobry, T., Y. N. Patt, and A. M. Despain. "Design Decisions Influencing the Microarchitecture for a Prolog Machine." *Proceedings of the MICRO-17 Conference.* 1984.
19. Tick, E., and D. Warren. "Towards a Pipelined Prolog Processor." *International Symposium on Logic Programming.* Atlantic City, 1984.
20. Van Roy, P. "A Prolog Compiler for the PLM." M.S. Thesis, University of California at Berkeley, CS Division, August 1984.
21. Dozier, H., T. Jones, F. Marshall, B. Wallace, and S. Wallach. "Super Supercomputer." *Computer Systems Equipment Design.* November 1984, pp. 17–22.
22. Convex Computer Corporation. *Convex Architecture Handbook.* Document No. 080–000120–000, Richardson, Tex., 1984.
23. Conery, J. S. "The AND/OR Process Model for Parallel Interpretation of Logic Programs." Ph.D. Dissertation, University of California at Irvine, Computer and Information Sciences Department, June 1983.
24. Chang, J. H., A. M. Despain, and D. DeGroot. "AND- Parallelism of Logic Programs Based on Static Data Dependency Analysis," memo University of California at Berkeley, CS Division, October 1984.
25. Chang, J. H., and A. M. Despain. "Semi-intelligent Backtracking of Prolog Based on a Static Data Dependency Analysis," submitted to Logic Programming Conference, 1985.
26. Franklin, M. A., D. F. Wann, and W. J. Thomas. "Pin Limitations and Partitioning of VLSI Interconnection Networks." *IEEE Transactions on Computers.* C–31–11 (1982), pp. 1109–1115.
27. Wann, D. F., and M. A. Franklin. "Asynchronous and Clocked Control Structures for VLSI-Based Interconnection Networks." *IEEE Transactions on Computers.* C–32–3 (1983), pp. 284–293.
28. Swan, R. J., S. H. Fuller, and D. P. Siewiorek. "Cm*—A Modular Multiprocessor." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 46), 1977, pp. 637–644.
29. Segall, Z., A. Singh, R. D. Snodgrass, A. K. Jones, and D. P. Siewiorek. "An Integrated Instrumentation Environment for Multiprocessors." *IEEE Transactions on Computers.* C–32–1 (1983), pp. 4–13.
30. Gajski, D., D. Kuck, D. Lawrie, and A. Sameh. "Cedar—A Large-scale Multiprocessor." *Proceedings of the International Conference on Parallel Processing.* Ann Arbor, Mich.: IEEE, 1983, pp. 524–529.
31. Dennis, J. B., G. A. Gao, and K. W. Todd. "Modeling the Weather with a Dataflow Supercomputer." *IEEE Transactions on Computers.* C–33–7 (1984), pp. 592–603.

A single chip bit-mapped image processor, MN8617

by KAZUFUMI SUZUKI, SHIGEO SHIMAZAKI, YUETSU OCHIAI, KAZUTOSHI IKEGAYA, ETSUKO HIROKAMI, KATSURA KAWAKAMI, and HIROAKI KOTERA
Matsushita Research Institute Tokyo, Inc.
Kawasaki, Japan

and

MIKIO FUJIWARA and YOSHIAKI KASUGA
Matsushita Electronics Corporation
Kyoto, Japan

ABSTRACT

The MN8617 is a special-purpose single-chip LSI processor designed for bit-mapped binary image manipulation in office information systems.

The chip, implemented on 2.5μm rule NMOS with 64,000 transistors, is characterized by its ability to access a 32 megabyte memory area and to scale images by 250 ns/pixel, along with other image-processing functions such as editing, drawing and analyzing. The high processing speed has been achieved by an integration of innovative parallel processing hardware and novel algorithms for scaling, and an implementation of dedicated microprogram-based machine instructions for image processing.

This paper primarily describes the algorithm and the special hardware for scaling.

INTRODUCTION

An advance in the bit-mapped image manipulation technique is now at the point where it can be applicable to image processing in office information systems. The high-speed bit-mapped image processor MN8617 has been developed for an application to the field.

The functions generally required for image processing in the office environment are as follows:

1. Drawing and editing
2. Pattern analysis and recognition: orthogonal transformation, filtering, and texture analysis
3. Improvement of image quality: dither, gamma correction, tone correction, color correction, and masking
4. Image generation: wire-frame, solid, fractal, and texture

Among these functions, drawing and editing are considered the most desired functions in the present office environment. The MN8617 is designed to implement these functions, with the additional capability of pattern analysis. The processor, accessible to a 32-megabyte memory area, performs the scaling at a speed of 250 ns/pixel and has versatile instructions for image processing such as drawing (lines, circles, ellipses, and painting-out), editing (translation, scaling, and rotation), and analysis (reading eight neighbors, logical filter, and contour trace).

DESIGN CONCEPTS

Three major points have been taken into account in the design of the processor; high-speed image transformation, high-quality image processing, and flexibility for configurating systems.

The high-speed transformation is achieved by a newly developed calculation algorithm that provides a maximum processing speed of 250 ns/pixel. The algorithm is implemented in an innovative hardware which occupies more than 30% of the whole chip area of the processor. The design effort is concentrated in implementation of image editing functions. It is considered that image editing functions are the most desired at present because they require a large amount of calculation and they are required very frequently in office information systems. Other functions for image recognition and image quality improvement, such as orthogonal transformation and color correction, are also useful for extracting information for recognition or for producing an improved image; however, these are considered a second priority of current office systems, although further developments are required.

Twenty-four-bit address lines are provided to the processor, which makes a 32-megabyte storage space accessible. This space is sufficient to contain 64 sheets of A4 size image with a resolution of 8 lines/mm. The processor is dedicated to binary image processing, being based on an assumption that half-tone images will be commonly constructed by binary pixels using the dither technique.

It is thought that the simplest way to increase the versatility of system configuration is to make the processor an auxiliary component of the host processor of a system. The MN8617 can be attached to the buses of a general-purpose processor. In other words, it processes information stored in a main memory of a host processor as bit-mapped image data, and it processes instructions of image transformation also stored in the main memory. The processor executes image processing according to its own instructions. This reduces the load of the host processor and increases the simplicity of image transformation programming. Because the instructions of the MN8617 are dedicated to image transformations, basic transformations such as expansion, contraction, and rotation can be programmed by a single instruction and several related parameters.

IMAGE PROCESSING MECHANISM

The image scaling of the MN8617 is performed by the special hardware Barrel Selector, based on the scaling algorithm using mapping patterns. Image data are stored in the memory on a bit-by-bit basis corresponding to the pixel. The memory is accessed on a word basis, which consists of 16 bits of image data or 16 pixels. A word of the image data is called the *image vector.*

Scaling Algorithm Using Mapping Patterns

The mapping patterns are binary patterns representing the rule of transformation from original images into resultant images. An original image vector and its elements are denoted as P and $P_1 P_2 \ldots Pj \ldots P_8$, respectively, as an 8-bit example. The resultant image vector and its elements are denoted as Q and $Q_1 Q_2 \ldots Qi \ldots Q_8$, respectively.

Assuming that the ratio of contraction is 4/13, the elements of the initial mapping pattern vector T are 10010010, as explained later.

In this case, as shown in Figure 1, the contracting process is performed by extracting the elements out of the original image vector P where each element in P corresponds to the unity element in the mapping pattern T. As a result, the elements of P corresponding to the elements of zero in T are eliminated.

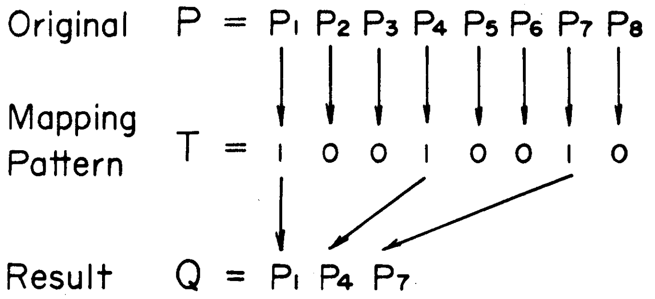


Figure 1—Image contraction using a mapping pattern

In the case of the expansion ratio of 13/4, as shown in Figure 2, the mapping pattern T is identical to the one used in the contraction of 4/13. However, the process of expansion differs from that of contraction and is performed sequentially for every pixel in the word. Assuming that Q_1 is assigned with P_1 at initial status, when i increases by 1 in the case when Ti equals zero, Qi is assigned to P_1 or to P_2 in the case when Ti equals unity. A similar assignment process is successively performed as the increment of i.

Parallel Processing Hardware Barrel Selector

The Barrel Selector is a parallel processing hardware for image scaling, which expands or contracts the original image according to the mapping patterns. Though there is a difference between the processes of expansion and contraction, as described previously, the Barrel Selector transforms both processes in the common hardware.

In the Barrel Selector, a group of selectors being controlled by the elements Ti of the mapping pattern T is arranged in a 16 × 16 matrix, which consists of 16 row gate circuits, as shown in Figure 3(a) and Figure 3(b). The vector of the original image P is transferred in the uppermost row and successively transferred to the lower row. As the vector moves down by a row, each element of the vector is processed. The vector Q of the resultant image is generated from the lowermost row.

In the expansion, the elements Pi of the original image are processed one by one for each row, starting from the element at the extreme left, as shown in Figure 4, as an 8-bit example. Each element Bi, j in the zone where $i > j$ takes an input datum from the above element $Bi - 1, j$. Each element Bi, j in the zone where $i \leqq j$ selects input datum either from $Bi - 1, j$ when Ti equals unity or from $Bi - 1, j - 1$ when Ti equals zero. When the processed image reaches to the bottom row, an expanded image composed of 8 bits is generated.

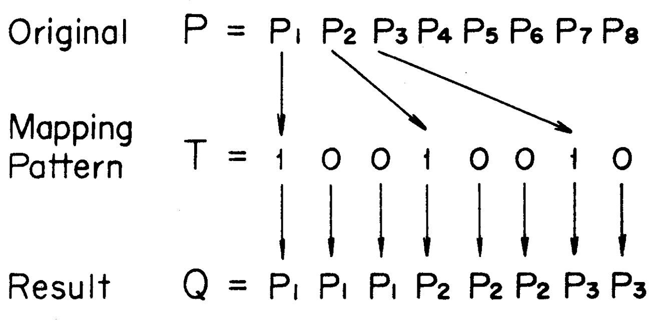


Figure 2—Image expansion using a mapping pattern

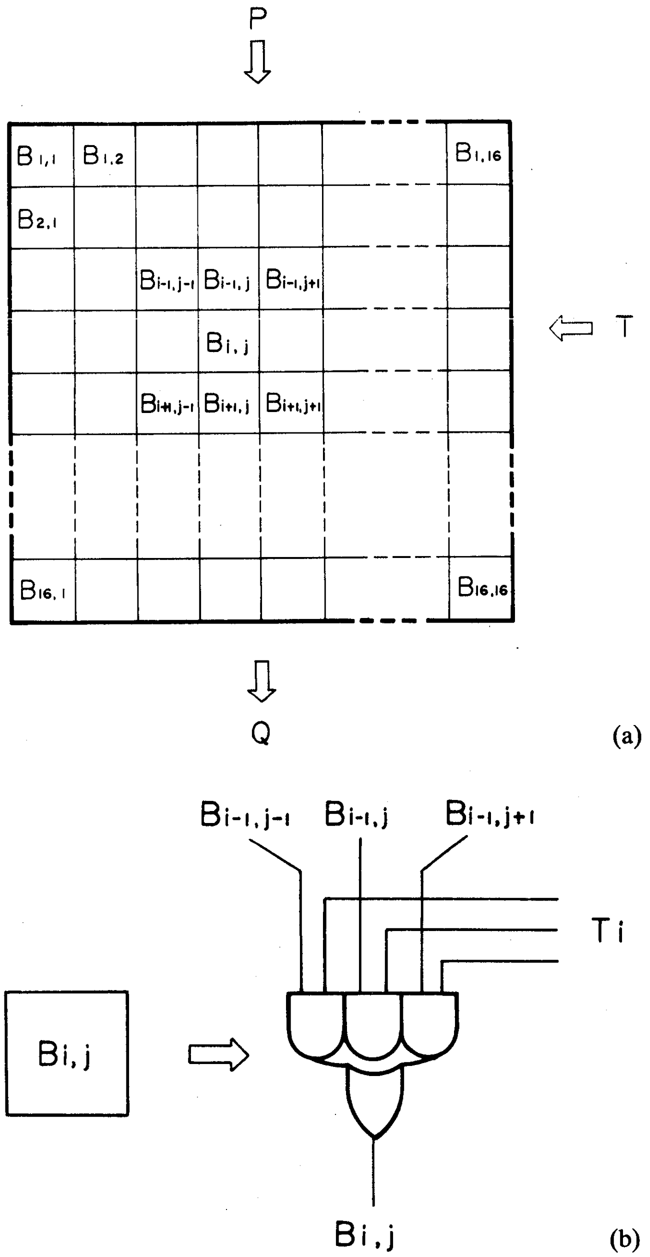


Figure 3—Barrel Selector: (a), structure of the Barrel Selector; (b), cell of the Barrel Selector

In the contraction, the element Pi of the original image is processed one by one for each row, starting from the element at the extreme right, as shown in Figure 5, also as an 8-bit example. Here, unlike the case of expansion, i is defined as the inverse direction. Each element Bi, j in the zone where $i > j$ takes an input datum from the above element $Bi + 1, j$. The element Bi, j in the zone where $i \leqq j$ selects an input datum either from $Bi + 1, j$ when Ti equals unity or from $Bi + 1, j + 1$ when Ti equals zero. When the original image reaches the bottom row, a contracted image composed of bits smaller than 8 bits is generated.

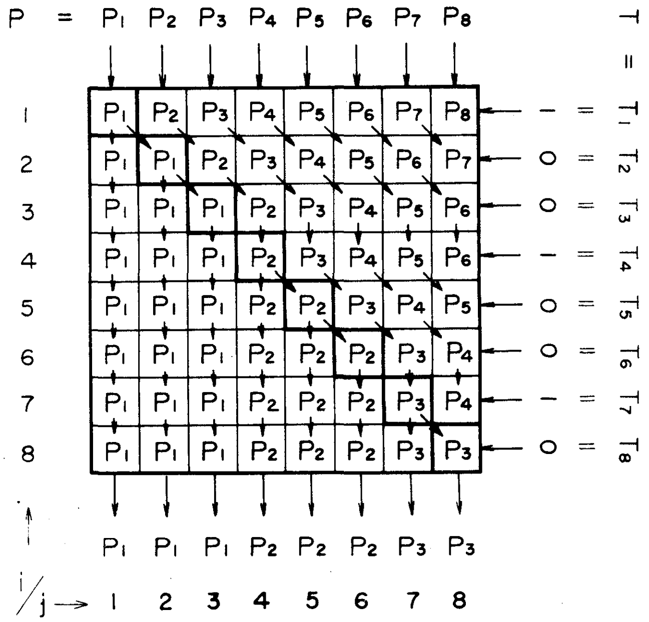

P = P1 P2 P3 P4 P5 P6 P7 P8
T =
1 P1 P2 P3 P4 P5 P6 P7 P8 1 = T1
2 P1 P1 P2 P3 P4 P5 P6 P7 0 = T2
3 P1 P1 P1 P2 P3 P4 P5 P6 0 = T3
4 P1 P1 P1 P2 P3 P4 P5 P6 1 = T4
5 P1 P1 P1 P2 P2 P3 P4 P5 0 = T5
6 P1 P1 P1 P2 P2 P2 P3 P4 0 = T6
7 P1 P1 P1 P2 P2 P2 P3 P4 1 = T7
8 P1 P1 P1 P2 P2 P2 P3 P3 0 = T8
P1 P1 P1 P2 P2 P2 P3 P3
i/j → 1 2 3 4 5 6 7 8

Figure 4—Image expansion in the Barrel Selector

Calculation of Mapping Patterns

The calculation of the mapping patterns in the MN8617 is executed by the microprogram, using an Extended DDA (Digital Differential Analyzer[1]) algorithm. The algorithm was developed originally for the previous image processor MN8614.[2,3]

Regarding the mapping from pixel *Pi* of the original image to pixel *Qj* of the resultant image, the relationship between *i* and *j* is expressed by the following equation, assuming that *M* and *N* are respectively a positive integer of mutually prime value with $M > N$, and N/M is the ratio of contraction and M/N of expansion.

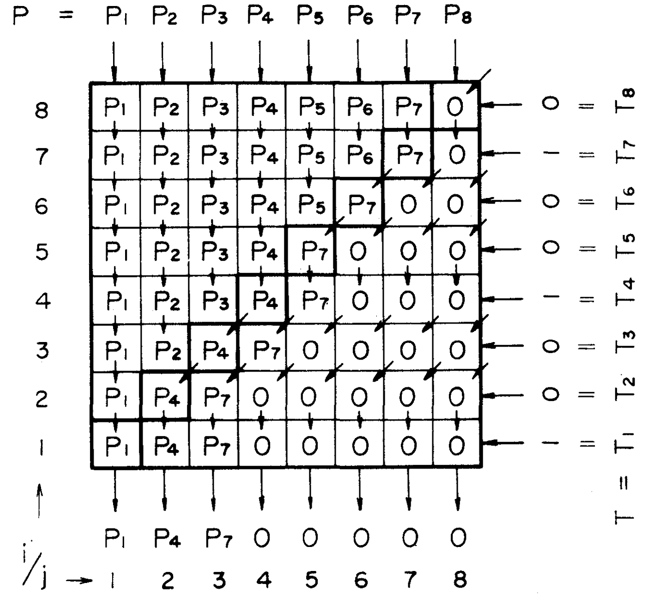

P = P1 P2 P3 P4 P5 P6 P7 P8
8 P1 P2 P3 P4 P5 P6 P7 0 0 = T8
7 P1 P2 P3 P4 P5 P6 P7 0 1 = T7
6 P1 P2 P3 P4 P5 P7 0 0 0 = T6
5 P1 P2 P3 P4 P7 0 0 0 0 = T5
4 P1 P2 P3 P4 P7 0 0 0 1 = T4
3 P1 P2 P4 P7 0 0 0 0 0 = T3
2 P1 P4 P7 0 0 0 0 0 0 = T2
1 P1 P4 P7 0 0 0 0 0 1 = T1
T =
P1 P4 P7 0 0 0 0 0
i/j → 1 2 3 4 5 6 7 8

Figure 5—Image contraction in the Barrel Selector

$$j = [(N * i + Ro)/M]$$

where the Gaussian notation $[X]$ represents the largest integer less than X. *Ro* is an initial remainder value of integer in the condition $0 \leqq Ro < M$.

The above calculation can be performed sequentially, without the need of multiplication or division, with the following algorithm. At every increment of i, the remainder R is replaced by the sum of R and the numerator N. Then, by comparing the R and the denominator M, the R is replaced by the difference of R and M, and j is replaced by the sum of j and unity when R is larger than or equal to M. R and j remain same when R is smaller than M. When R is replaced, a carry-over is generated; otherwise it is reset to zero. The procedure can be summarized as follows:

```
$i \leftarrow i + 1$
$R \leftarrow R + N$
$R \leftarrow R - M$,  $j \leftarrow j + 1$  and  carry $\leftarrow 1$  if $R \geqq M$
```

Figure 6 shows an example of calculation of the mapping pattern when $N/M = 4/13$ and $Ro = 12$. In this example, the elements of the mapping pattern T are represented by *Ti*. When i is increased, *Ti* equals unity at the points where j changes, and it equals zero otherwise. In other words, *Ti* equals unity at the point where the carryover occurs in the calculation, and zero otherwise. *Ti* is also derived directly by the following expression, using multiplication and division:

$$\begin{aligned} \mathrm{Ti} &= j(i) - j(i-1) \\ &= [(N * i + Ro)/M] - [(N * (i-1) + Ro)/M] \end{aligned}$$

where [ ] denotes as Gaussian notation.

The mapping patterns are binary patterns with a period of

i	$R_{i-1}+N=R_i \rightarrow R_i-\left[\frac{R}{M}\right]\cdot M$	j	T_i	
1	12 + 4 = 16 → 3	1	1	
2	3 + 4 = 7	1	0	
3	7 + 4 = 11	1	0	
4	11 + 4 = 15 → 2	2	1	
5	2 + 4 = 6	2	0	T
6	6 + 4 = 10	2	0	
7	10 + 4 = 14 → 1	3	1	
8	1 + 4 = 5	3	0	
9	5 + 4 = 9	3	0	
10	9 + 4 = 13 → 0	4	1	
11	0 + 4 = 4	4	0	T
12	4 + 4 = 8	4	0	
13	8 + 4 = 12	4	0	

Figure 6—Calculation of mapping pattern

M, and its length is generally longer than 16 bits. The MN8617 has the stack registers with 16 words (256 bits); the words of the mapping patterns are read out one by one and processed subsequently. If the mapping patterns are longer than 16 words, the remainder *R* of the sixteenth processing is reserved, and the next processing is performed using the remainder.

Configuration of the Scaling Unit

The scaling unit is composed of Input Registers (IH, IL), an Input Barrel Shifter, a Barrel Selector, an Output Barrel Shifter, Output Registers (OH, OL), and a Mapping Pattern Register (MPR), as shown in Figure 7. The unit processes scaling at a high speed under the control of the lower-level microprogram. Because of the existence of two Barrel Shifters, the operations are performed without distinction between the word-boundaries and bit-boundaries. The original image vectors being input from the bus to the Input Registers are moved to the Barrel Selector through the Input Barrel Shifter. After the scaling, these are stored in the Output Registers through the Output Barrel Shifter.

Two Input Registers are implemented in order to achieve smooth calculation in the processor and efficient data transfer between the processor and image memory. The queuing of two 16-bit registers makes it possible to avoid a pause in calculation at word boundaries. The Input Barrel Shifter increases the speed of the bit-boundary processing in the source image area by extracting an arbitrary piece of consecutive 16-bit image data from the Input Registers and transferring it to the Barrel Selector.

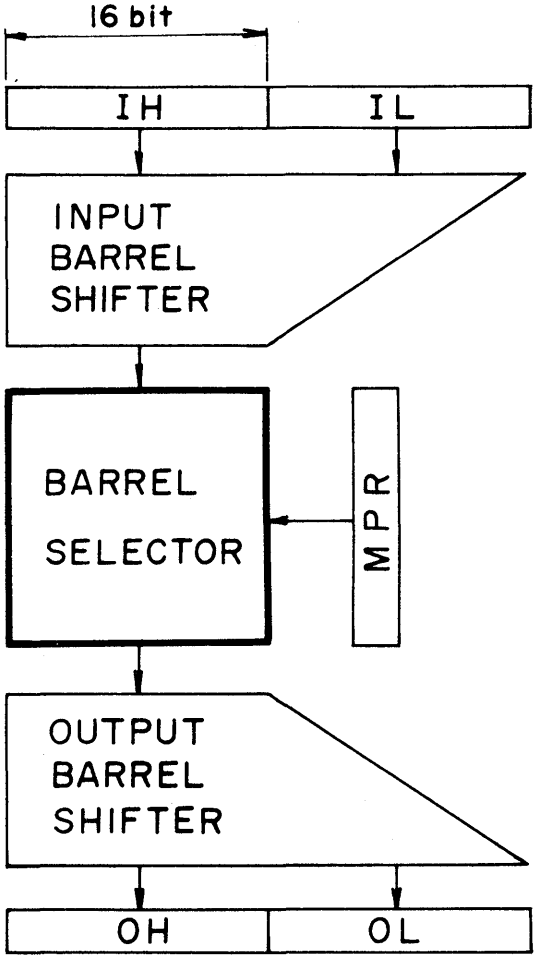


Figure 7—A block diagram of scaling unit

The Barrel Selector, being controlled by sets of 16-bit binary mapping patterns, performs scaling of the input image vector in parallel, as described above.

The Output Barrel Shifter increases the speed of the bit-boundary processing for the destination image area by shifting the generated resultant vector by an arbitrary number of bits within 16 bits and placing the vector at an arbitrary location on the two Output Registers. The two Output Registers are also incorporated, for the same reason as the Input Registers.

Each process is performed with a single mapping pattern vector stored in the 16-word mapping pattern stack. Each vector is read out and stored in the Mapping Pattern Register (MPR), for the purpose of controlling the Barrel Selector.

ARCHITECTURE

The MN8617 can be connected to the buses of general-purpose microprocessor systems and constructs multiprocessor systems. Therefore, without changing the memory structure of the existing system, its image-processing capabilities will be reinforced easily, and any functions not existing in the MN8617 will be made up by the host processor.

A block diagram of the processor is shown in Figure 8. The area on the left side of the diagram, including the Scaling Unit and the Mapping Pattern Stack, shows the hardwares added to the previously described original processor. The machine instructions are fetched from the memory space, which is commonly used for the program and the image, and are led to the Interpret Unit through the Instruction Queue and the Instruction Register. The Interpret Unit, controlled by the upper-level microprogram, issues the starting addresses of the lower-level microroutines, which are located in the Control Unit. The 16-bit ALU, General Registers, Image Processing Unit, and other units are controlled by the lower-level microprogram. Eight out of 24 general registers are assigned as explicitly referenced in the machine instructions. The remaining 16 are used as working registers, which include the buffers and the registers to hold the scaling ratio and others. The EAR (Extended Address Register), WAR (Word Address Register), and BPR (Bit Position Register) are called current position registers, which designate one bit out of 256 megabits of image memories. The width of image area is defined by the WDR (Width Defining Register). PR (Pattern Register) is assigned the routine of drawing lines or painting-out. The resultant status of operations is held in the STR (STatus Register).

The machine instructions of the MN8617 are divided into two categories. The first is the dedicated set of instructions for image processing, which consists of 79 instructions. The second is the general set of instructions for program flow control and arithmetic operations, consisting of 62 instructions which

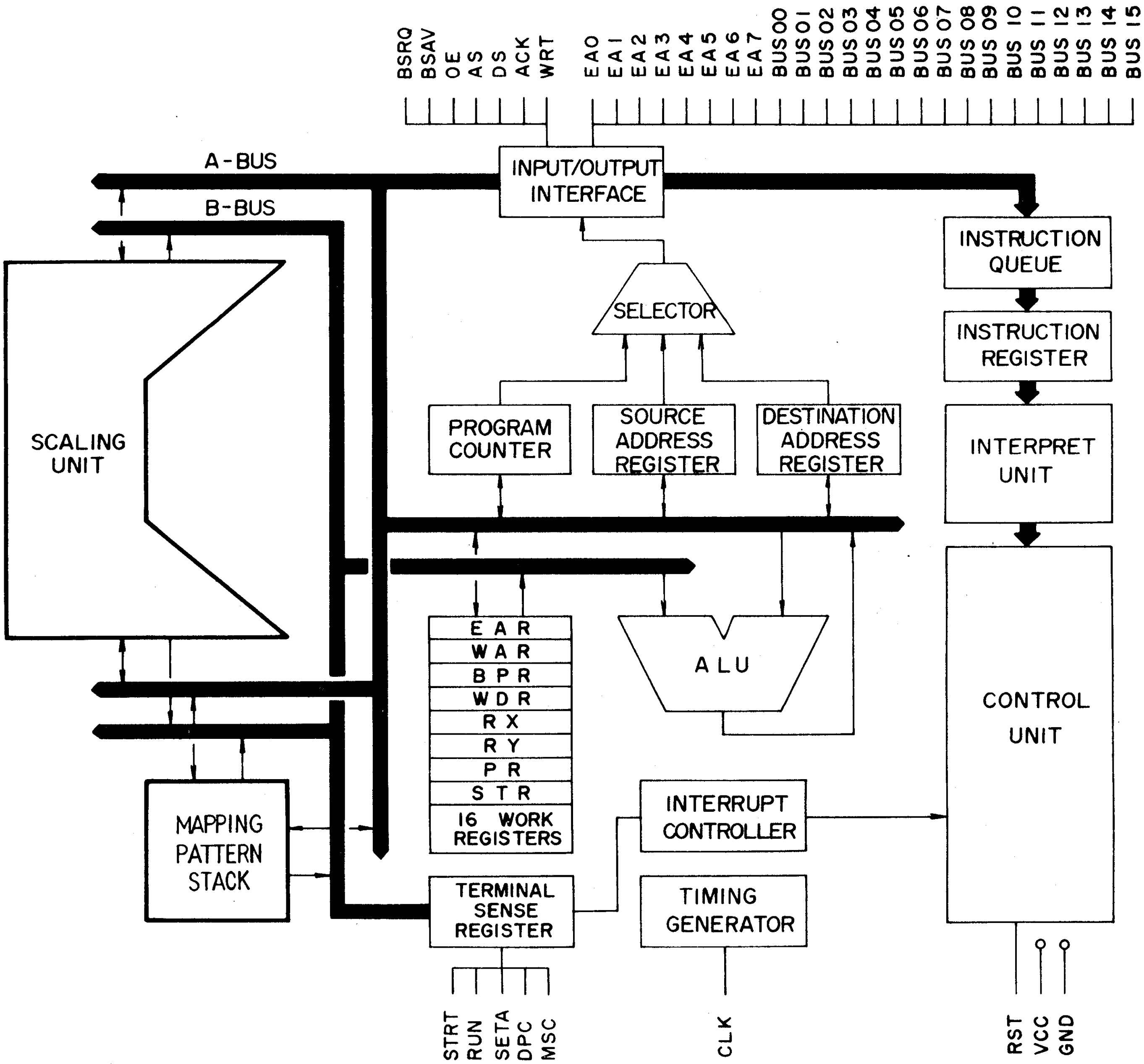


Figure 8—A block diagram of MN8617

is similar to general-purpose processors. In the upper-level microprogram, consisting of 100 words, the first and second sets of instructions are 34% and 66%, respectively. As for the lower-level microprogram, consisting of 1400 words, the first set is 88% and the second set is 12%.

The chip is packaged in a 40-pin DIL. A sample of image scaling using the MN8617 is shown in Figure 9.

CONCLUSION

The MN8617 adopts newly developed parallel processing hardware using the mapping pattern algorithm, which can edit bit-mapped image at the maximum speed of 250 ns/pixel. The processing speed of the MN8617 is 20 times faster than its predecessor, MN8614, as shown in Table I. For practical applications to image-processing systems using the MN8617, a multiprocessor construction can be easily configured with the host processor to improve performance.

Although further improvement in the processing speed and the development of an algorithm for additional image-processing functions suitable to the LSI environment are needed, the state-of-the-art processor MN8617 is expected to enhance image-handling capabilities in office information systems.

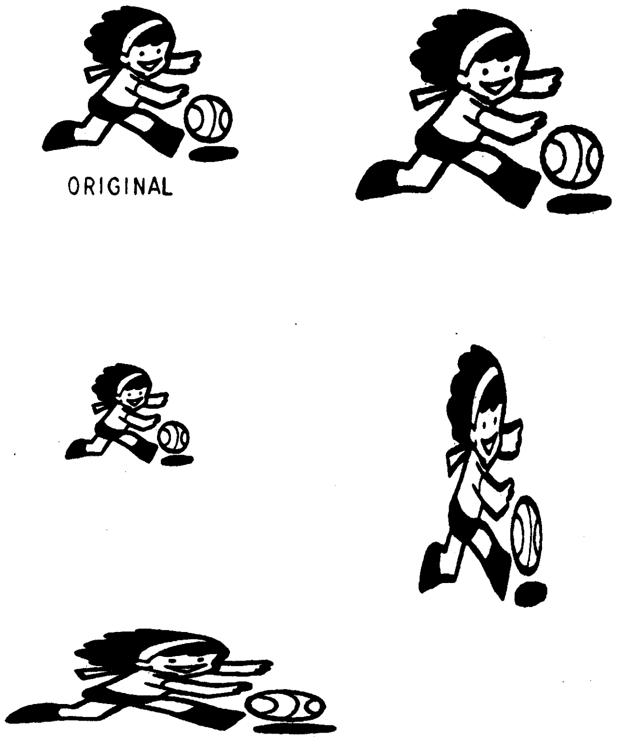


Figure 9—A sample of image editing using MN8617

TABLE I—Comparison of specifications

Item (Unit)	MN8617	MN8614
Accessible Memory Area (megabyte)	32	0.5
Instructions (number)		
Image Processing	79	31
Program Control	62	35
General Registers (word)	24	16
Execution Time (us/element)		
Drawing		
Line	4.3	5.2
Circle, Ellipse	5.8	6.9
Painting-out (rectangle)	0.07	0.2
Painting-out (arbitrary shape)	0.1	1.9
Editing		
Translation	0.11	0.53
Expansion	0.2	3.3
Contraction	0.24	3.2
Rotation (arbitrary angle)	6.0	7.8
Rotation (90-degree)	0.48	6.4
Analysis		
Read eight neighbor	5.7	—
Logical filter	5.4	—
Contour trace	20.0	—
Clock (MHz)	15.0	13.3
Package (pin-DIL)	40	40

ACKNOWLEDGMENT

The authors would like to thank Mr. T. Tsumura of Panafacom Co., Ltd., and Dr. I. Morishita of Tokyo University for their suggestions and encouragement, and to thank Dr. T. Kawasaki of Matsushita Electric Corporation for his promotion of the project. They also thank Mr. T. Nakamura of Matsushita Research Institute Tokyo, Inc., for his critical reading and comments on the manuscript.

REFERENCES

1. Newman, W. M., and R. F. Sproul. *Principle of Interactive Computer Graphics* (2nd ed.). New York: McGraw-Hill, 1981, pp. 22–28.
2. Kawakami, K., and S. Shimazaki. "A Special Purpose LSI Processor using the DDA Algorithm for Image Transformation." *IEEE 1984 Proceedings of the 11th Annual International Symposium on Computer Architecture,* pp. 48–54.
3. "MN8614 Logical Image Controller Hardware Manual." Matsushita Electronics Corporation, Kyoto, Japan, 1983.

Dataflow architectures for knowledge representation systems*

by LUBOMIR BIC
University of California
Irvine, California

ABSTRACT

Most AI systems today are written in languages based on the sequential von Neumann model of computation and thus are not well suited to parallel processing. In this paper we propose to use logic programming as a bridge between the high-level operations performed by knowledge representation systems based on semantic nets and highly-parallel computer architectures. It will be shown how knowledge recorded in semantic nets, including the corresponding information retrieval operations, may be expressed in terms of logic programming. Such representation may be processed in a data-driven manner and thus permits computer architectures consisting of very large numbers of independent processing elements to be exploited.

* This work was supported by the NSF Grant MCS-8117516: The UCI Dataflow Databases Project.

INTRODUCTION

Most AI system today are written in languages that are based on the sequential von Neumann model of computation and thus are executed on single-processor control-driven computer architectures. Even in the case of multiprocessors, the underlying model is still based on the basic principle of a fetch-execute cycle repeated by a processing element according to an instruction counter. The large semantic gap between the high-level operations performed by an AI system, e.g. logical inferences, pattern matches, or the searching of large spaces, and the typical repertoire of low-level machine instructions into which such operations must be mapped, is a serious limitation to high performance.

In recent years, logic programming[1,2] has gained considerable attention in the realm of AI research. One of the main attractions of this approach is that logic programs do not presuppose a sequential von Neumann architecture and are therefore well suited to parallel processing. In this paper we propose to use logic programming as a bridge between AI systems based on the notion of semantic nets and computer architecture. We will first discuss the correspondence between semantic nets and logic programming. Then, by presenting a data-driven model for the execution of logic programs, we introduce a model for processing of semantic nets, suitable for execution on a computer architecture consisting of a very large number of homogeneous processing elements.

SEMANTIC NETWORKS

The use of semantic networks is a well-known approach to representation of knowledge in the realm of artificial intelligence. Even though there is not a precise, universally accepted definition of semantic nets, their common characteristics and advantages have been discussed extensively in the literature.[3,4] A semantic net consists of a collection of nodes interconnected by arcs. Each node represents an object (physical object, set, relationship, event, logical entity) and each arc represents a binary relation. Usually, there is a finite set of such relations, e.g., "subset off", "element off", "agent", "consequence".

The main attraction of semantic nets largely centers around two factors. First, they provide sufficient expressive power to encode facts or concepts that are encodable in any other formal system. Second, the structures constituting a net serve themselves as a guide for information retrieval: information most closely related to a node is found simply by following arcs emanating from that node.

The main problem with semantic nets is the lack of exact rules or procedures governing the retrieval and modification of information represented by the semantic net. Most systems based on semantic nets assume the existence of some *outside* agent—a collection of programs—that embodies the knowledge about how to use the information in the net. This assumption makes parallel processing of semantic nets extremely difficult; it requires that the central processing agent divides computation into smaller tasks, distributes these to different processing elements, and coordinates their progress. As in other multiprocessor systems based on the notion of centralized control, these tasks are extremely difficult to solve and usually permit just a few processing elements to be exploited.

To solve the above problems, we propose a different model of computation: instead of assuming an outside agent to manipulate and use the network, the processing is incorporated into the network itself, i.e., each node is assumed to have some computational power and the ability to communicate with its neighbors via the network's arcs. This point of view will permit the mapping of a semantic net onto an architecture consisting of large numbers of independent processing elements without the need for any centralized control.

SEMANTIC NETWORKS AND LOGIC PROGRAMMING

Deliyanni and Kowalski[5] have proposed an extended form of semantic nets which may be mapped onto the firm grounds of first order predicate logic and thus provides an exact formalisms for processing such nets. The following paragraphs describe briefly the correspondence between logic programming and the extended semantic nets.

A logic program is a set of *clauses* of the form

$$p_0 \leftarrow p_1, \ldots, p_n.$$

Each p_i is called a *literal* and has the form $p(t_1, \ldots, t_m)$, where p is a *predicate* symbol and $t_1, \ldots, t_m$ are *terms*. Terms may be constants, variables, or functors. p_0 is called the *head* or *conclusion,* and p_1 through p_n form the *body* or *conditions* of the clause.

A clause with an empty set of conditions is called an *assertion* and is used to represent *explicit* facts. If we restrict all predicates to be binary, each literal $p(t_1, t_2)$ may be represented in the form of a directed arc as shown in Figure 1. The

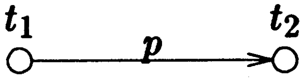


Figure 1—An arc corresponding to the predicate $p(t_1, t_2)$

arrow head is used to record the order in which the terms of the literal were given. This information must be preserved when the literal represents an asymmetric relation. (As will be discussed later, the arrow heads do not prescribe the direction in which tokens may flow through the graph.)

The collection of all assertions then may be represented as a graph. Literals sharing the same term result in arcs connected to one another via the corresponding node. Since many terms may be shared among different literals, graphs of arbitrary complexity may result. This is illustrated by the sample logic program shown below (adopted from Reference 5):

1. *act(e, give)* ←
2. *obj(e, book)* ←
3. *actor(e, john)* ←
4. *rec(e, mary)* ←
5. *likes(john, book)* ←
6. *likes(X, Z)* ← *act(U, give), obj(U, Y), actor(U, X), rec(U, Z), likes(X, Y)*
7. ← *likes(john, mary)*

Lines 1 through 5 are assertions which form the semantic net shown in Figure 2.* The constant *e* represents an act or event of giving, where *john* is the actor, *mary* is the recipient, and *book* is the object being given. The fact that *john* likes the *book* he is giving to *mary* is also recorded (line 5).

A clause which contains both a head and a body can be interpreted as recording *implicit* information. For example, line 6 of the above logic program can be paraphrased as follows: if it can be shown that *X* gives an object *Y* to *Z* (more formally, there exists an *act U* of giving, of which *Y* is the object, *X* is the actor, *Z* is the recipient), and *X* likes the object *Y*, then it may be concluded that *X* likes the person *Z*. To represent such statements graphically, Deliyanni and Kowalski introduce two types of arcs—one to represent conditions and another to represent conclusions. With such a representation, deductive information, which is normally part of the programs which process the semantic net, becomes actually part of the network itself.

A clause with an empty conclusion is interpreted as a request or *goal* which the system tries to solve through the principle of *resolution*—a sequence of unification steps attempting to derive an empty clause. In Figure 1, line 7 shows a goal that may be paraphrased as the query: "does *john* like *mary*?" To find an answer, the system will try to unify this goal with some other clause, in our example, the clause on line 6. The variables *X* and *Z* are bound to the constants *john* and *mary*, respectively, thus generating five new goals—*act(U, give)*, *obj(U, Y)*, *actor(U, john)*, *rec(U, mary)*, and *likes(john, Y)*—all of which must be solved in order to satisfy the original goal. Figure 3 shows the graphical representation of the above five goals. The resolution process is repeated until one (or more) solutions are found, or no further unifications are possible.

In the graphical representation, the body of each clause may be viewed as a *graph template*. Each unification within the resolution process then corresponds to the following *graph matching* problem: determine possible bindings of all free variables in the given graph template to constants such that the template matches some portion of the underlying semantic net. In the above example, binding the variables *U*, and *Y* to the constants *e*, and *book*, respectively, yields a match between the template of Figure 3 and the semantic net of Figure 2, implying that the predicate *likes(john, mary)* is true.

DATAFLOW VIEW OF PROCESSING

In the approach described in the previous section, the processing of the (extended) semantic networks is defined in terms of executing logic programs. That is, the resolution principle of logic programming is used to extract knowledge from the semantic net. Graphically, each unification step corresponds to finding a match between the graph template representing the given clause and some portion of the underlying semantic net.[6,7]

The model presented in this paper is based on the idea of performing the unification steps by directly operating on graphs, instead of executing the corresponding logic programs. The necessary graph matching may be performed in a data-driven manner,[8] which lends itself to an implementation on a highly parallel computer architecture. The basic idea is to view the semantic net as a *dataflow graph*. That is, each node of the network is not simply a passive element of information; rather, it is an *active* element, capable of receiving,

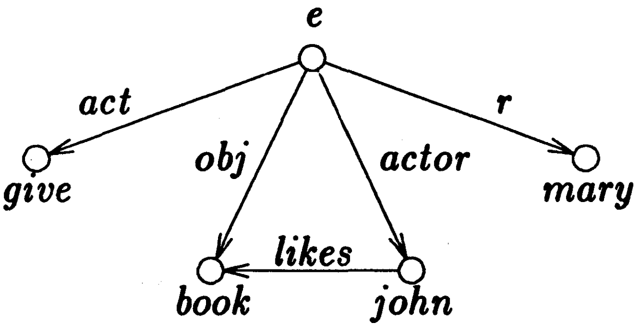


Figure 2—A semantic net representing the act of giving

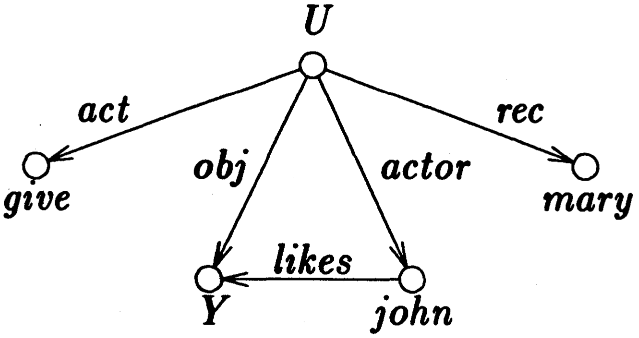


Figure 3—A graph template representing the query 'does *john* like *mary*?'

* Throughout this paper, lower case letters are used to denote constants while capitals are used to denote free variables.

processing, and emitting value tokens traveling asynchronously along the network's arcs. The necessary graph matching is then accomplished by placing the graph template on tokens and injecting these into specific nodes of the semantic net. From their injection points, tokens replicate simultaneously into many possible directions thus searching for the given pattern. Since many processing elements may be engaged in the processing and replication of tokens, a computer architecture consisting of a large number of independent processing elements may be employed.

DATA-DRIVEN GRAPH MATCHING

In this section we develop the procedures for propagating the template-carrying tokens through the underlying semantic net. (These procedures are based on those presented originally in Reference 6.)

Matching of a Single Arc

We consider first the simplest form of a template, consisting of a single arc $T_1 \xrightarrow{p} T_2$. This arc corresponds to the literal $p(T_1, T_2)$. To find a match, we need to determine possible bindings for the terms T_1 and T_2 such that the graph template $T_1 \xrightarrow{p} T_2$ matches some are of the semantic net. Operationally, this is accomplished by placing the graph template on a token and injecting it into specific nodes of the semantic net. From each of these nodes the token is replicated along existing arcs in an attempt to find a match. We can distinguish the following four cases:

1. Both nodes T_1 and T_2 are bound to round terms t_1 and t_2, respectively. Since there can be only one occurrence of each of the nodes t_1 and t_2 in the semantic net, the token is injected into one of these, say t_1. This node then replicates the token along all arcs labeled p that emanate from t_1. If one of the nodes receiving the replicated token matches the second term t_2, the subgoal is solved successfully; otherwise there is no direct match for this template. The same result is obtained when the token is initially injected into t_2 from which it replicates in a search for t_1. This will be denoted by reversing the direction of the arc: $T_2 \xleftarrow{p} T_1$.
2. The node T_1 is bound to a ground term t_1 while the node T_2 is a free variable. As in the first case, the token is injected into the node t_1 from which it is replicated along all arcs labeled p. This time, however, any node t_2 receiving the replicated token may be bound to the variable T_2 and hence presents a solution to the given subgoal $p(T_1, T_2)$.
3. The node T_2 is bound to a ground term t_2 while the node T_1 is free. In this case, reversing the arc to $T_2 \xleftarrow{p} T_1$ yields a situation analogous to (b), where the first term is bound while the second is free. Hence the same approach can be taken.
4. Both variables T_1 and T_2 are free. This case differs from the previous three in that there is no unique injection point for the token. Rather any node of the semantic net is a potential binding for either variable and hence the token must be broadcast to *all* nodes of the semantic net. Each of these nodes binds the first variable T_1 to its own content and replicates the token along all arcs labeled p in the same way as described under (b). In other words, the search is started in all nodes simultaneously.

Matching of Linear Chains of Arcs

In this section we extend the scheme for matching of individual arcs as discussed above to cope with sequences of arcs of the form:

$$p_1(T_1^1, T_2^1), p_2(T_1^2, T_2^2), \ldots, p_n(T_1^n, T_2^n),$$

where each T_1^i must satisfy one of the following restrictions:

1. It must match the term T_2^{i-1}, or
2. It must be bound to a constant.

In the first case the arcs corresponding to the two adjacent literals p_i and p_{i-1} are connected, resulting in a graph template of the following form:

$$\ldots \xrightarrow{p_{i-1}} T_1^i \xrightarrow{p_i} \ldots.$$

In the second case, where T_1^i is bound to a constant, we introduce a dummy arc, denoted as $\Rightarrow$, to create an artificial connection between the two adjacent arcs p_i and p_{i-1}, i.e.,

$$\ldots \xrightarrow{p_{i-1}} T^{i-1} \Rightarrow T_1^i \xrightarrow{p_i} \ldots.$$

A sequence of literals where each T_1^i satisfies one of the two above restrictions has the following important property: all literals in that sequence, except possibly the first, will have the left-most term T_1^i bound when the sequence is processed from left to right. We will refer to such a sequence as *linear*.

The fitting of a linear sequence into the semantic net then corresponds to the following graph fitting problem: determine possible bindings for all terms T_j^i such that the graph template matches some path in the semantic net. Operationally, this is accomplished as follows: A token, carrying the entire graph template, is injected into nodes of the semantic net that may be bound to the first term T_1^1. (As was the case with individual subgoals, only one such node t_1 will exist if T_1^1 is bound to a ground term; otherwise the token must be injected into all nodes of the semantic net.) Each node t_1 receiving the injected token will replicate it along all arcs that match the template arc p_1. Each of the nodes t_2, receiving the replicated token, will attempt to bind itself to T_2^1. At this point, the next arc of the template is either a regular arc p_2 or it is a dummy arc $\Rightarrow$. In the first case, t_2 would continue the propagation of the token along all arcs matching the name p_2. In the second case, the token is simply sent to the node that matches the corresponding term T_1^2. There is at most one such node since, by definition, T_1^2 must be bound.

The same steps are performed by all nodes t_i receiving the token, which results in a stepwise expansion of the graph template into all possible directions of the semantic net. Each branch continues to grow until one of the following conditions occur:

a. A node t_i is unable to bind itself to the corresponding node T_2^i (i.e., T_2^i is already bound to a term different from t_i), or, no arc emanating from t_i matches the corresponding template arc p_i. This indicates that no match can be found along this path.
b. The last node T_2^n of the template has been reached, implying the detection of a match for the given graph template.

Matching of Tree-Structured Graph Templates

We define a tree–structured template recursively as follows:

1. A linear chain of arcs

   $$p_1(T_1^1, T_2^1), p_2(T_1^2, T_2^2), \ldots, p_n(T_1^n, T_2^n),$$

   as defined in the previous section is a tree.

2. If $p_{i1}(T_1^i, \ldots), \ldots$ and $p_{i2}(T_1^i, \ldots), \ldots$ are trees, then the construct

   $$(p_{i1}(T_1^i, \ldots), \ldots)(p_{i2}(T_1^i, \ldots), \ldots)$$

forms a tree, where T_1^i, which is shared by both literals p_{i1} and p_{i2}, becomes the root of that tree. Note that by repeatedly applying the above rules, arbitrarily many subtrees may be created at each node.

The algorithm for fitting linear graph templates described in the previous section may be extended to cope with tree-structured templates as follows:

Assume that the node t_i receives a tree-structured template of the form

$$(p_{i1}(T_1^i, \ldots), \ldots)(p_{i2}(T_1^i, \ldots), \ldots) \ldots (p_{in}(T_1^i, \ldots), \ldots)$$

As a first step, t_i will attempt to bind itself to the root variable T_1^i. If this is successful, it will start a matching process for each of the n subtrees. That is, for each subtree $p_{ij}(T_1^i \ldots), \ldots$, the node t_i forms a new token and replicates it along all arcs that match the name p_{ij}. As was the case with linear templates, it will await responses from all directions into which tokens have been sent. Solutions for the original tree-structured template are then constructed by combining the possible solutions for each of the individual subtrees.

To illustrate this procedure, consider again the semantic net of Figure 2. Assume, however, that instead of the template shown in Figure 3, we are interested in finding matches for the tree–structured template of Figure 7. The fitting process will begin with injecting a token carrying the given template into the node of the semantic net that matches the template's root node; this is the node *john*. From there, a copy of the template is replicated along all arcs matching the label *actor*. All receiving nodes will bind their own names to the variable U; in the above example, this will be the node *e*. At this point there are three subtrees, all of which must be fitted into the semantic net. The first will be sent along all arcs matching the label *act*; the receiving node *give* will match successfully against the corresponding template node *give*. The second subtree is sent along arcs matching the label *rec,* thus reaching the node *mary*. Finally, the third subtree is sent along all arcs that match the label *obj,* and, since the template variable Y may be bound to the node *book,* it continues along the arc *likes* back to the node *john.* Since all subtrees have successfully been fitted into the semantic net, the question "does *john* like *mary?*" has been answered affirmatively.

Dynamic Graph Templates

It is not always possible or convenient to express the requests for extracting some knowledge from a semantic network as a fixed graph template. To illustrate such a situation, consider Figure 4, which shows several concept nodes arranged into a "*is-a*" hierarchy. A property called *color* is associated with the node *typical-cat.*

Typical for such a hierarchy is the principle of *property inheritance,* which states that a property applies not only to the node to which it is attached but also to all descendants of that node within the hierarchy. For example, the property *color* in Figure 4 applies not only to the node *typical-cat* but also to the individuals *bertha* and *fritz*. In logic programming, this can be expressed conveniently as the following recursive clauses:

$$\textit{inherited–prop}(X, Y)\!: \leftarrow \textit{loc–prop}(X, Y)$$

$$\textit{inherited–prop}(X, Y)\!: \leftarrow \textit{is–a}(X, Z), \textit{inherited–prop}(Z, Y)$$

Operationally, this program may be paraphrased as follows: Check first if Y is X's *local* property. If so, the query is satisfied, else go to X's parent in the '*is–a*' hierarchy and perform the same check. Repeat this process until the query is satisfied or the top of the hierarchy is reached, indicating a failure.

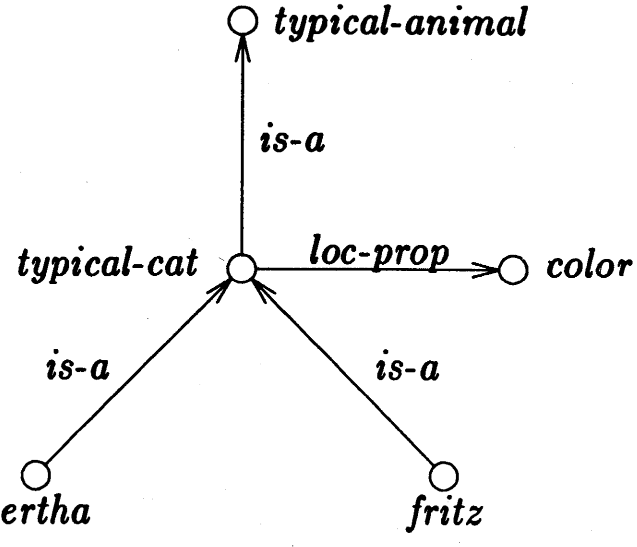


Figure 4—An 'is-a' hierarchy

By binding actual node names to one or both of the variables X and Y, the semantics of the query can be changed. For example, the meaning of the query *inherited–prop*(*fritz, color*) is interpreted as "does fritz have a color", and yields the value true or false. The meaning of *inherited–prop*(*fritz, Y*), on the other hand, is to "find all properties of fritz"; finally, the meaning of *inherited–prop*(*X, color*) is to "find all elements that have color as one of their properties". The body of each of the above queries may itself be viewed as a network (or a collection of networks). Consider for example the query *inherited–prop*(*fritz, color*). From the above recursive definition of *inherited–prop* we obtain

inherited–prop(*fritz, color*): ← loc–prop(*fritz, color*)

inherited–prop(*fritz, color*): ← *is–a*(*fritz, Z*),
  inherited–prop(*Z, color*)

which can be represented as an infinite sequence of networks shown in Figure 5. The task of answering that query is then to determine whether at least one of these networks matches some portion of the underlying database network. As indicated earlier, this may be performed by asynchronous propagation of tokens through the database network.

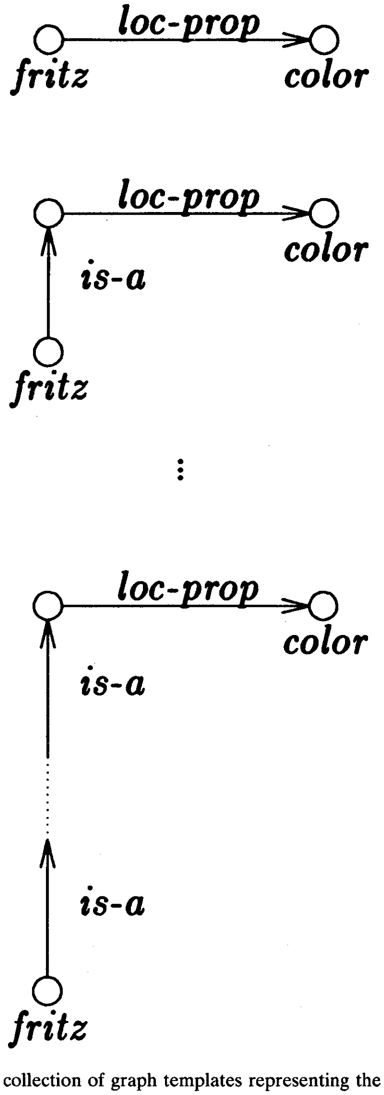


Figure 5—A collection of graph templates representing the query 'does fritz have a color attribute?'

In the above simple case a query token is propagated from *fritz* to nodes at higher levels in the 'is–a' hierarchy and each such node checks for the existence of a *color* at its own level. If no such node is found when the top of the hierarchy is reached, the query reports a failure; otherwise a positive answer is given.

TRANSFORMATION OF GRAPH TEMPLATES

The procedures for graph fitting presented in the previous section assume that all templates have the form of a tree. (A liner chain is a special case of a tree.) In the most general case, however, graph templates of arbitrary interconnection topology may occur. Since tokens in dataflow systems propagate asynchronously, the detection of cyclic patterns is difficult. To solve this problem, each template is first transformed into a tree–form, for which token-propagation procedures have already been developed.

There is a number of ways to transform a given graph template into a tree. One possible approach is to find a depth first spanning tree, which partitions the template into two sets of edges—*tree* edges and *back* edges. (A procedure for constructing such spanning trees may be found, for example, in[9].) Figure 6 shows one possible spanning tree constructed for the template of Figure 3; back edges are shown as dotted lines. By replicating the nodes at which back edges terminate, we obtain the desired tree structure shown in Figure 7. At each node of the template, the corresponding subtrees represent independent goals which can potentially be processed in parallel since no free variables are shared among subtrees.

ARCHITECTURAL REQUIREMENTS

According to the model described in previous sections, a semantic net is a dataflow graph from which information is extracted by an asynchronous propagation of tokens. In this

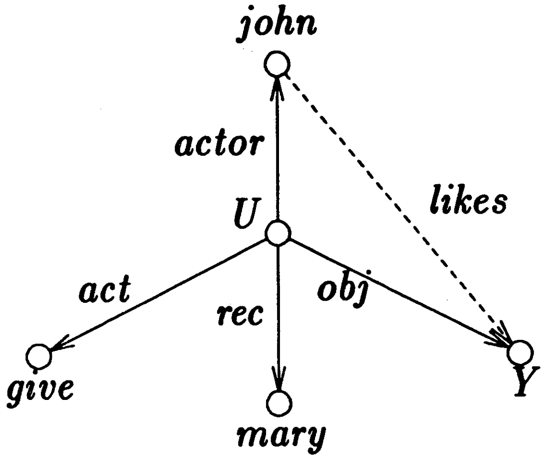


Figure 6—A spanning tree derived from the template of Figure 5

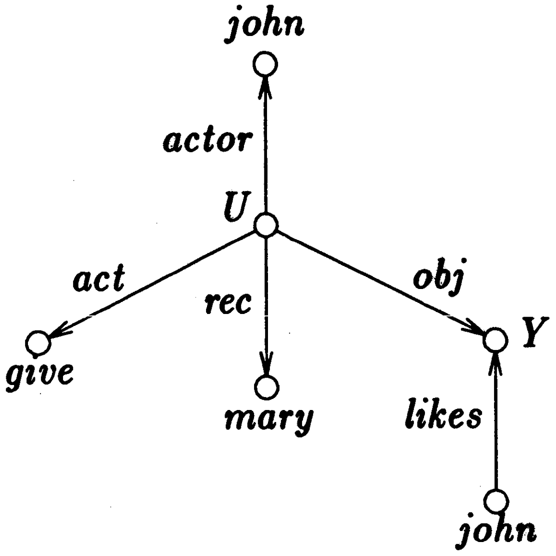


Figure 7—A tree-structured graph template

section we examine the architectural requirements necessary to support this model.

Implementation of the Semantic Net

According to the model, each node of the semantic net is active, i.e., it must be capable of exchanging tokens with other nodes and of performing some computation on the received tokens. This implies the following requirements:

- Each node must be mapped onto a processing element (PE) which will perform the necessary token processing and communication operations on the node's behalf; thus the PE is the incarnation of all nodes mapped onto that PE. There is no centralized control to start or supervise the execution of individual PEs; rather, the operation of each PE is triggered by the arrival of a token.
- Nodes must be able to communicate via logical connections corresponding to the arcs of the semantic net. This implies that any two PEs holding nodes connected via a logical arc must be able to communicate with one another; the communication may be direct or via other PEs.

The mapping of nodes constituting the semantic net onto a computer architecture consisting of p processing elements, may be accomplished by providing a global mapping function f. Given a graph node n, f yields a number $f(n)$, ranging between 1 and p, which designates the PE that will hold the node n during execution. This mapping may be done statically or dynamically.

The lack of centralized control permits potentially a very large number of PEs to be employed since scheduling is performed implicitly by the arrival of tokens. The issue, however, is to provide the necessary communication network to permit the nodes of the semantic net to exchange tokens. Since the topology of the semantic net will not be known a priory, and may also change during execution, we must assume that each PE is capable of communicating (directly or indirectly) with *any* other PE. Unfortunately, since the number of PEs is expected to be large, a fully connected network will not be feasible; rather, the choice of a suitable interconnection scheme is a trade-off between an acceptable rate of token flow and the cost of the necessary hardware. (Simulation experiments are currently being conducted in our laboratory to investigate this issue.)

Assuming that a communication path may be established between any two PEs, a logical arc between two nodes $n1$ and $n2$ of the semantic net may be represented by recording the name of the node $n1$ with the node $n2$, and vice versa. To send a token along such an arc is then accomplished by finding a physical path between the PEs holding the two nodes and routing the token toward its destination.

Injection of Tokens

According to the model it must be possible to inject tokens into any node of the semantic net. In the implementation it implies finding the address of the PE holding the receiving node. This is very similar to the problem of providing communication channels (logical arcs) between nodes as discussed above and thus may be solved using the same mechanisms: a sender, which in the case of token injection would be an external source, as opposed to one of the PEs, can determine the PE that holds the destination node by applying to it the same mapping function f that has been used initially to map the graph onto the architecture. This PE number is carried by the token as its destination address when it is being routed through the physical network.

CONCLUSIONS

The main objective of the research presented in this paper was to find a bridge between the high-level operations performed to retrieve knowledge from a semantic net, and the underlying computer architecture. We have shown that the processing of semantic nets may be expressed in terms of logic programming, which in turn lends itself to non-von Neumann style of computation. The model is based on an asynchronous propagation of tokens through the semantic net in a search for graph patterns corresponding to given queries. Since a large number of independent processing elements may be engaged in the processing and forwarding of individual tokens, a highly parallel computer architecture may be employed. It should be noted that the architectural requirements of the model are very similar to those of general dataflow systems, which suggests that existing architectures, designed to execute dataflow programs, may easily be adapted to support the proposed model and thus provide a highly parallel execution environment for the processing of semantic nets.

REFERENCES

1. van Emden, M. H., and R. A. Kowalski. "The Semantics of Predicate Logic as a Programming Language." *Journal of the ACM, 23* (1976), p. 4.
2. Kowalski, R. A. "Logic Programming for the Fifth Generation." *Proceedings of the International Conference on Fifth Generation Systems,* London, U.K.: SPL Internationals, 1982.
3. Findler, N. V. (Ed.). *Associative Networks.* New York: Academic Press, 1979.
4. *Computer.* Special Issue on Knowledge Representation, *16* (1983), p. 10.
5. Deliyanni, A., and R. A. Kowalski. "Logic and Semantic Networks." *Communications of the ACM, 22* (1979), p. 2.
6. Bic, L. "A Data-Driven Model for Parallel Interpretation of Logic Programs." *Proceedings of the International Conference on Fifth Generation Computer Systems,* Tokyo, Japan: Institute for New Generation Computer Technology, 1984.
7. Bic, L. "Execution of Logic Programs on Dataflow Architectures." *Proceedings of the 11th International Symposium on Computer Architecture,* Los Alamitos, Calif.: IEEE Computer Society, 1984.
8. *Computer.* Special Issue on Dataflow Systems, *15* (1982), 2.
9. Aho, A. V., J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Reading, Mass.: Addison-Wesley, 1976.

An overview of the *DADO* parallel computer

by MARK D. LERNER, GERALD Q. MAGUIRE JR., and SALVATORE J. STOLFO
Columbia University
New York, New York

ABSTRACT

DADO is a special purpose parallel computer designed for the rapid execution of artificial intelligence expert systems. This article discusses the *DADO* hardware and software systems, with emphasis on the question of *granularity*. *DADO* is designed as a fine-grain machine constructed from many thousands of processing elements (PEs) interconnected in a complete binary tree. Two prototype systems, *DADO1* and *DADO2,* are detailed. Each PE of these prototypes consists of a commercially available microprocessor chip, memory chips, and an additional semicustom I/O processor designed at Columbia University. The software includes a kernel and parallel languages. Under development are several artificial intelligence systems, including a production system interpreter, a logic programming language, and an expert system building tool.

INTRODUCTION

A considerable amount of interest has recently been generated in specialized machine architectures designed for the very rapid execution of Artificial Intelligence (AI) software. The Japanese Fifth Generation Machine Project, for example, promises to deliver a functioning device capable of computing solutions of large PROLOG programs at execution rates on the order of many thousands to perhaps millions of *logical inferences per second* (LIPS). Such a device will require high-speed hardware executing a large number of primitive symbol manipulation tasks many times faster than today's fastest computers. This rather ambitious goal has led some researchers to suspect that a fundamentally different computer organization is necessary to achieve this performance. Thus, *parallel processing* has assumed an important position in current AI research.

Since 1980, we have been exploring various parallel processing architectures suitable for the rapid execution of AI software implemented in production system (PS) and logic programming (PROLOG, for example) form.[1,2,3,4] A number of parallel algorithms have been developed and studied, each designed to capture the inherent parallelism in a wide range of symbol manipulation tasks. This study has led to the development of a specific machine architecture which has come to be called *DADO*.[5,6,7]

DADO is a binary tree-structured multiprocessor architecture incorporating thousands of moderately powerful processing elements (PEs). Each PE consists of a fully programmable microcomputer with a modest amount of local memory (in the range of 16 to 20 Kbytes, in the second prototype) and a specialized I/O chip designed to accelerate inter-PE communication. A full-scale production version of the machine may comprise many thousands of processors implemented in VLSI technology.

Other parallel tree-structured machines have been proposed in the literature for accelerating a wide range of applications from database operations to more general applications involving numerical operations.[8,9] *DADO* distinguishes itself from these other architectures in several ways.

First, *DADO* is not fully detailed. That is, although we have settled upon the notion of large-scale parallelism by incorporating thousands of PEs interconnected in a complete binary tree, the granularity (storage capacity and functionality) of each PE remains an open theoretical and important practical issue. We are thus hedging our bets by remaining uncommitted to any specific PE design. By studying "real-world" applications executed on a *DADO* prototype, our proposed architectural studies will shed more light on the granularity suitable for a production version of the machine.

Second, *DADO* is designed for a specialized set of applications implemented with the AI production system and logic programming formalisms.[3] Thus, we can expect *DADO* to provide important information for other researchers interested in accelerating AI computation.

Third, the execution modes of a *DADO* PE are rather unique. Each PE may operate in Single Instruction Multiple Data (SIMD) mode[10] whereby instructions are executed as broadcast by some ancestor PE in the tree. Alternatively, a PE may operate in Multiple Instruction Multiple Data (MIMD) mode by executing instructions from its local RAM. Such a PE may, however, broadcast instructions for execution by descendant PEs in SIMD mode. This rather simple architectural principle allows *DADO* to be fully partitioned into a number of distinct "sub-*DADO*s," each executing a distinct task. Thus, *DADO* may potentially implement a variety of parallel algorithms spanning a vertical spectrum from the small grain operations typical of the parallel manipulation of databases up to more coarse-grain activities associated with multitasking. Indeed, it is our experience that this flexible architectural design provides a rather powerful "semantic-base" allowing relatively easy programming of these various parallel activities.

Fourth, our current design of the second prototype machine, *DADO2,* employs two physical binary tree interconnections. This provides fault tolerance as described in a later section.

Fifth, we have designed *DADO* around commercially available technology rather than designing everything from scratch. In addition, the software has been designed with portability in mind (such as changing the type of microprocessor). Thus, it is quite possible that future *DADO* machines could rapidly be implemented and directly employed in real-world settings much more quickly than other proposed machines waiting for technology to catch up with their architectural needs.

In a number of previous papers,[3,11,12,13,14,15,16,17,18,19] we have detailed the hardware design of the *DADO* machine, as well as the software systems that have been implemented to date and currently under development. The algorithms that were invented for the parallel execution of production systems and logic programs were not sufficient alone for deciding several specific hardware design issues. The decision was made, therefore, to build functioning prototype systems that provide the means to carry out empirical studies. Towards that end, it was necessary to develop concurrently both hardware and software systems that implement and demonstrate the most important architectural principles of the machine while providing the necessary laboratory environment for experimentation and study.

In the following sections, we detail the status of the *DADO1* and *DADO2* prototype hardware and software systems. Several kinds of software systems are under development. Low-level system software includes the kernel and application language support resident in each *DADO* PE. The parallel languages are PPLM,[13] and a newly developed high-level parallel LISP (PPSL).[17,19] Artificial Intelligence (AI) software consists of several very-high-level systems: OPS5,[20] HerbAl,[21] and LPS[16,18] languages. Knowledge acquisition software (such as DTEX[22]) uses these layers to develop a working expert system which is then available to users.

HARDWARE

The hardware configuration of *DADO* consists of a large number of processing elements (PEs) connected by a tree-structured bus. Each PE in the prototype systems consists of an Intel 8751 microprocessor with 4 Kbytes of onchip EPROM, 128 bytes of onchip RAM, 16 Kbytes of RAM with parity, and an I/O processor. The initial *DADO1* prototype PE was built without the I/O processor, which is simulated by a software kernel for parallel communication resident in each EPROM.

The I/O processor provides very fast bidirectional communication and selection of data. It also supports both a *global interrupt* and *context switch.* This allows any processor to interrupt the entire machine and place it into debug mode. When an interrupt occurs, the host processor is given control. It can then read and write the individual processor memories as part of the debugging process. Subsequently, the machine can be restarted without loss of any information.

As noted, the PEs are interconnected as a binary tree. The root PE is connected to two descendant PEs, which in turn are connected to two descendants each. This interconnection topology was selected for two reasons, (1) It can be very efficiently implemented in VLSI technology, and (2) this architectural configuration is well suited for a large class of problems.

A prototype of the hardware, *DADO1,* has been operational since April 1983. It has been used to develop further hardware and software. Using gate array technology, we have designed a specialized I/O processor for the *DADO2* PE to accelerate inter-PE communication. These I/O chips are interconnected in a binary tree. Each microcomputer chip in a *DADO2* PE is also interconnected in a second complete binary tree. These two tree interconnections provide a measure of fault tolerance. Thus, if a *DADO2* I/O chip fails, the processor connections will allow the machine to continue operating, as evidenced by the small 15 element *DADO1* prototype which employs processor interconnections exclusively. Alternatively, if a *DADO2* processor fails, the I/O chip interconnection will allow easy fault isolation and diagnosis. Software for reconfiguration and fault tolerance, however, is not yet developed and not part of the current effort.

Work is largely complete on the larger *DADO2* machine. This computer will be configured with 1023 PEs and 16 megabytes of RAM. The construction of this machine involved the design and integration of the I/O processor; the connection circuitry in the form of circuit boards and a backplane; the power supply and cabinet; and a high speed I/O interface to a conventional host.

Communication

DADO hardware prototyping began three years ago in 1981. At that time, the Intel 8751 was the only commercially available single chip microcomputer in existence that provided 4 parallel 8 bit ports. These ports allowed us to conveniently interconnect a number of PEs directly without additional logic or chips. The 4K onchip EPROM also allowed us to conveniently implement software with the opportunity to reprogram the ROM as the software evolved.

If we were to initiate prototyping today, we would use a 32 bit two address microcomputer chip. Such a design is expected to immediately deliver a 16 fold speed advantage over the 8751.

Synchronization between processors was originally implemented with a four-cycle handshake protocol, implemented in software. This is being replaced with a semicustom I/O processor, which is expected to provide substantial performance improvements for two reasons; (1) the I/O chip can transfer a byte of data throughout the entire machine in the time needed for a typical microprocessor instruction, (2) the microprocessor is no longer burdened with the communication task.

The I/O chip performs in hardware three main parallel operations: (1) flow of information from one processor to its descendants (broadcast), (2) the selection of the largest (or smallest) value of a set distributed throughout the machine (max/min resolve), and (3) the flow of information from descendants towards their ancestors (report). In addition, the I/O processor provides the hardware to help monitor and debug large software systems (global interrupt and context switch), as well as memory support.

The I/O processor was completely designed at Columbia University. We used sophisticated CAD/CAM tools in this phase. A VALID ScaldSystem served in all phases of development: from schematic entry using a hierarchical methodology, through rough simulation, and then board layout. The design was verified at LSI Logic with their LDS simulator, and subsequently fabricated using approximately 1850 gates of HCMOS gate array.

Circuit Boards

The extraordinarily regular layout of the binary tree made it possible to design relatively simple circuit boards. Most of the interconnections are within the PEs, between a RAM chip and its processor. These wires are shorter than 1 inch in length. The remaining lines are for communication and clock signals. These interconnections are very regular. Indeed, all 32 boards are completely interchangeable.

The *DADO* circuit board was designed using two CAD tools; the VALID ScaldSystem and an Advanced Electronic Design (AED) workstation running Caesar CAD software. The VALID was used to create a drawing of the board. We developed software on the VALID to produce a complete holelist from this drawing. The holelist indicates the positions of all drill holes, as well as the endpoints of the connecting wires. Hierarchical design features of the VALID allowed us to specify the interconnections once and then replicate them as needed.

The AED workstation was used to the design artwork which describes the power, ground, and silkscreen layers of the circuit board. We subsequently used software to verify that the holelist agreed with the artwork. The DARPA MOSIS silicon foundry then produced full-sized transparencies of the three layers. The actual board fabrication was done by Multiwire Corporation, using the artwork and holelist. The motherboard was also designed with the AED station, and has been fabricated by MOSIS.

Packaging

To minimize the volume of the machine we chose to place 32 processors on each board and to maintain a very low profile on each board. To achieve these goals, we used two techniques: (1) The I/O processor is packaged in a 64 pin PGA package which occupies 1 square inch of surface area. We exploited this square package to develop an efficient component layout. (2) Components are placed into eyelets. These occupy less height above the board than the traditional sockets, while providing the excellent mechanical and electrical qualities that are required for a new machine.

Each of the 32 boards houses 32 processors, 32 I/O processors, and 512 Kbytes of RAM with parity. The resulting hardware is simpler than a DEC VAX-11/750, and has a cumulative processing power in excess of 570 MIPS.

Interface to Control Processor

The *DADO* machine functions as an attached processor controlled by a conventional computer such as a DEC VAX or AT&T 3B machine. The conventional computer serves as a control processor (CP). *DADO* communicates with the CP in two ways: either an RS232 connection at 9600 baud, or with a parallel interface. The latter allows us to use the direct memory access (DMA) mode of a DEC DR11W. To achieve large data transfer rates (approximately 500 Kbytes/second) we designed a ROM-based microcoded interface processor. This handles the protocol needed to transfer data to and from *DADO*.

SYSTEM SOFTWARE

The system software includes a kernel (described below) and two parallel languages developed at Columbia. Copies of the kernel are located at each PE. This coordinates use of the hardware and provides essential services that are critical to the operation of the machine, or are very frequently required by the high level languages. The various layers of software are shown in Figure 1.

The kernel supports the most fundamental operations of the *DADO* machine. This includes control of the hardware lines, synchronization of processors, and the transfer of data. The kernel functions are detailed in Reference 2.

This kernel was extensively used without modification for over a year. Minor modifications have been made to support the I/O chip of *DADO2*. Recently, the kernel has been restructured to provide the resources needed for the PPSL language described below.

The lowest level of the kernel actually manipulates logic levels; an intermediate level implements protocols with these signals. The highest level is accessible to the user, and provides the following basic functions:

1. *Broadcast* to send information to the descendant processors.
2. *Resolve* to select one processor from a candidate set.

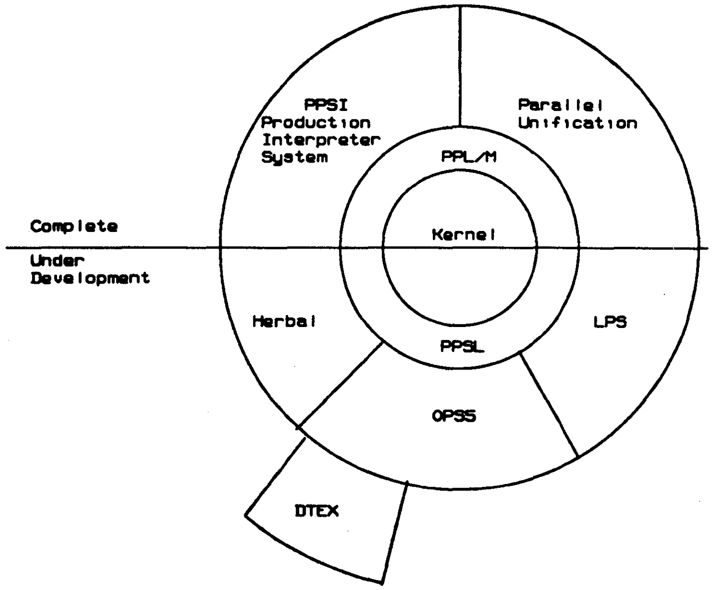


Figure 1—Layers of system software

3. *Report* to send information to the root processor.
4. *Send* to send data to a neighbor.
5. *Receive* to receive data from a neighbor.
6. *Enable* and *Disable* to make processors execute the instructions or to prevent them from doing anything except forward data.
7. *MIMD* and *Sync* to partition the tree into independently executing subtrees, and to resynchronize.

PPLM

Parallel programming requires the programmer to provide two kinds of specifications: (1) A control structure for the coordination of PEs. This may be viewed as a set of conditions that determine when a code segment should be executed. (2) The sequential code segments to be executed in particular processors. Communication and coordination between these segments may either be explicitly stated in the program, or implicit from the context in which they are used.

The first *DADO* language implemented, PPLM, uses explicit statements to synchronize the processors. PPLM provides direct access to the SIMD, MIMD, and communication features of the machine. Interspersed in the program are statements to be executed in parallel. These are designated via a syntactic construct, the DO SIMD block.

PPLM[13] translates source programs into Intel's PLM-51[23] language. PPLM checks for syntax errors, and in addition enforces the semantics needed to prevent deadlocks in a parallel computer. For example, if a processor expects to receive information from its parent, then the parent must place the information on the broadcast bus. The compiler determines whether these conditions are satisfied and generates an error message if they are not.

This program is currently executed on a VAX running UNIX. The translated output is sent to an Intel MDS with Kermit* and then compiled with Intel's PLM-51 compiler. PPLM is implemented via the standard UNIX tools (C, Lex, and Yacc) and can be easily transported to a variety of machines.

PPSL

The second language implemented is PPSL.[17] This is a LISP system which has been extended to permit the specification of parallel control. The most salient feature of the language is the declaration of functions as MIMD or SIMD, with semantic rules to determine how these forms are executed in *DADO*.

The language provides the ability to manipulate arbitrary LISP expressions (s-expressions, or forms) throughout the tree. The language constructs include:

1. (BROADCAST form1 form2), in which *form2* is evaluated in one processor and then stored into descendant processors as determined by their independent evaluations of *form1*.
2. (SETQSLICE form1 form2), in which *form2* is evaluated in each descendant processor and stored into the *form1* of each processor.
3. (DOMIMD form), in which designated descendant processors disconnect themselves from the tree-structured broadcast bus, and begin execution of *form*.

The PPSL language can be executed on a conventional computer by use of a *DADO* simulator. This simplifies software development. Several programs have been written using this tool, including an OPS5 implementation, a parallel sort program, a spelling corrector, and numerous production system algorithms.

The PPSL language has the ability to *relocate* s-expressions (the fundamental LISP data structure). This is required to communicate the s-expressions between processors and the host. Storage structures have to be relocated when moved between processors. Moreover, communication of arbitrary length items is required for the language, and the language supports a protocol that is both robust and efficient.

The PPSL constructs are implemented primarily in LISP. The programmer need not be concerned with low level details. Instead, routines such as Broadcast, SetqSlice, and DoMimd provide implicit communication. In addition, the PPSL programmer has access to the complete *DADO* kernel (this may be needed, for example, in applications that utilize non-LISP segments of code such as assembler).

Static scoping is being implemented with a program analysis tool. This is a scope-checker, which will perform a static analysis to determine if scope rules are violated.

Compilation-time error checking and optimization will be essential. Diagnosis at compile time saves substantial effort at later phases. Common errors include misuse of scoping and use of the wrong type of data objects. (See Figure 2 for PPLM and PPSL code.)

LISP Compiler

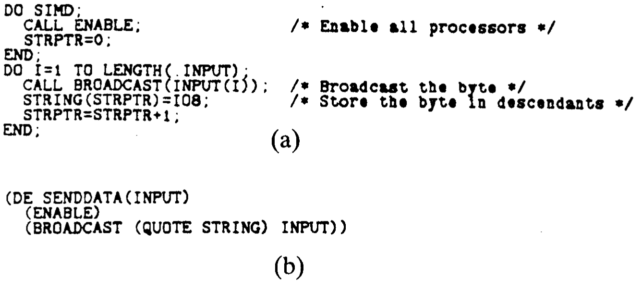
A LISP compiler[19] based on other works[24,25] has been developed for the *DADO* prototype machines. It translates LISP into assembly language for the Intel 8751 processor, which in turn is assembled and linked with commercially available tools.† This language can be used to program *DADO;* moreover, the high-level PPSL language runs on *DADO* by use of this underlying LISP structure.

In addition to the bare compiler, we have developed a runtime library and other software to support programming in PPSL. The most important parts of the work are a *kernel interface* and *garbage collector.* Other significant aspects include independent compilation, optimizations, interprocessor communication of s-expressions, and several kinds of diagnostics.

Independent compilation allows us to create object code libraries. The original design includes provision for the independent compilation by use of a linkage editor and a global symbol file.

* Kermit is a widely used file transfer and terminal emulation program developed at the Columbia University Center for Computing Activities.

† We use the Nuvatec Uniware assembler and linkeditor.



```
DO SIMD;
  CALL ENABLE;                    /* Enable all processors */
  STRPTR=0;
END;
DO I=1 TO LENGTH( INPUT);
  CALL BROADCAST(INPUT(I));       /* Broadcast the byte */
  STRING(STRPTR)=I08;             /* Store the byte in descendants */
  STRPTR=STRPTR+1;
END;
```

(a)

```
(DE SENDDATA(INPUT)
  (ENABLE)
  (BROADCAST (QUOTE STRING) INPUT))
```

(b)

Figure 2—(a) PPLM code to store the string contained in INPUT; (b) PPSL code to store the string contained in INPUT

We are currently investigating optimization techniques to reduce code size. We have developed a preliminary data-compaction algorithm. This shows that code improvements are possible by two techniques. First, the algorithm is used as an analysis tool to show where compiler internals must be changed to improve the code. Second is common substring detection followed by automatic subroutine generation. Preliminary studies indicate that up to 40% compaction is possible.

The compiler will eventually provide direct support for parallel debugging. It will support four techniques:

1. Read out code or data from the tree to compare with that stored on the host
2. Break points, to gain control of a PE when it executes certain code
3. Rebinding of functions, to modify what a function does
4. Profiling, to determine how frequently functions are used

These will be incorporated into a graphics-based debugging system.

Graphics

We are currently using high resolution color graphics to debug and demonstrate programs. We have designed a graphics based debugging system that draws a picture of a binary tree. By use of two physical screens, we will be able to probe into processing elements as required. These graphic routines are used to demonstrate the operation of *DADO.*

ARTIFICIAL INTELLIGENCE SOFTWARE

Several artificial intelligence areas are being studied as application areas for *DADO*. We have made significant progress in three of these: a production system language (HerbAl), a logic programming language (LPS), and a generic expert system tool (DTEX). Although *DADO* was originally designed to be a production system machine, the additional uses attest to the flexibility of the architecture design.

A production system[26,27] consists of a number of production rules (PM), which are matched against a database of facts called *working memory* (WM). Rules have a left hand side (condition) and a right hand side (action). The system repeatedly executes the following cycle:

1. Match: Determine those productions with conditions that are true given the current state of the working memory database.
2. Select: choose one of the true productions.
3. Execute: Execute the actions specified by the right hand side of the production. Actions include additions, deletions, and modifications to the working memory.

Although production systems have been put to practical use on sequential machines with considerable success, there is a critical need for significantly faster execution speeds. We have demonstrated that production systems can be executed by use of parallel hardware, and expected to achieve dramatic performance improvements in the near future.

One major question in the design of the parallel hardware is the *granularity* question. This is concerned with the resources (functionality, and storage capacity) available at each processing element. When the size of RAM is increased in a coarse-grain approach, more rules and WM elements may be stored and processed by an individual PE. Since fewer PEs are available, however, less work may be performed in parallel. Conversely, by restricting the size of RAM in a finer-grain approach, fewer rules and WM elements may be located at a PE, but the additional PEs can possibly perform more operations in parallel.

Recent statistics reported for R1* indicate that of a total of 2000 rules and several hundred WM elements, on average, 30 to 50 rules need to be matched against WM on each cycle of operation. Thus, even if 2000 finer-grain PEs were available to process the rules (say, one rule per processor), only 30 to 50 PEs would perform useful work on each cycle of execution. If we used 30 to 50 coarser-grain processors storing many more rules, instead, all of the inherent production matching parallelism would be captured, making more effective use of the hardware.

This approach ignores the advantages of processing WM elements in parallel. In a manner analogous to partitioning rules to a set of PEs, WM elements may also be distributed to a set of independent PEs distinct from those storing rules.[4,28] The grain size of a PE may then directly affect the number of WM elements that may be processed concurrently. Thus, with a larger number of smaller PEs, WM may be operated upon more efficiently than with a smaller number of larger PEs. It follows that a "tug-of-war" between production-level and WM-level parallelism provides an interesting theoretical arena to study the tradeoffs involved between parallel processors of varying granularity.

A larger question has now arisen. The reported statistics for R1 are based on a problem-solving formalism that has been fine-tuned for fast execution on serial processors, namely OPS5. *Thus, the inherent parallelism in R1 may bear little resemblance to the inherent parallelism in the problem R1 solves; but it is, in our opinion, an artifact of current OPS5*

* R1 is an expert system written in OPS5 that configures DEC VAX computers.

production system programming on serial machines. The OPS5 implementation of R1 provides little information about what subproblems are inherently parallelizable. Indeed, all subproblems are carefully handcrafted to be sequentially executed using "control elements." Our aim is to provide other formalisms that allow one to explore and implement much more parallelism than OPS5 encodes or encourages.

For example, two simple experiments have been performed on two small production system programs running at Columbia. A few additional parallel constructs were added to OPS5 which simulate the parallel manipulation of WM[29] and multiple rule applications on each cycle of execution. The statistics indicate that these slight enhancements to OPS5 produce a *factor of 6–10 fewer* production system cycles of execution to solve the same problem. Similar studies are underway using the ACE expert system.[30,31] Thus, we can expect that a formalism slightly different than OPS5 will admit much more opportunity for parallelism and concomitant speed-up.

Original DADO *PS Algorithm*

The original algorithm, implemented in the PPLM language, works as follows. The *DADO* machine is divided into subtrees, in which there is a distinguished level for production memory (PM-level) and a lower level for working memory (WM-level). During the match phase, each PM-level PE enters MIMD mode and uses its SIMD PEs as a content-addressable memory. No state is saved between cycles because the algorithm exploits large-scale parallelism in the match phase.

During the act phase, changes to WM are enforced when the winning production reports the actions to the *DADO* root. These actions are subsequently broadcast to all PM-level PEs, which in turn make the appropriate updates to their descendant PEs' storage.

One characteristic of production systems is temporal redundancy. Few changes are made to WM on each cycle, which restricts in practice the number of possible rule matchings on subsequent cycles. The above algorithm recomputes all matches before each production cycle, and does not save any state information between cycles. Thus, it does not exploit temporal redundancy. On the other hand, the OPS5 production system[32] saves state between production cycles and never recomputes comparisons. This exploits temporal redundancy, yet there may be excessive overhead, particularly when large changes are needed to WM.

The Treat Algorithm

A new algorithm, Temporally *RE*dundant *A*ssociative *T*ree algorithm (TREAT)[33] uses ideas from both of the above algorithms. It maintains the most useful state information, while less useful comparisons are recomputed as needed. This approach is based upon several observations.

When a new element is added to the WM, every rule that becomes eligible for execution will also contain the new WM element. A technique used by OPS5 (called alpha-memories) allows us to quickly compute matches for the new WM element. Similarly, when WM is deleted, the tree is examined in parallel and all conflicts are removed concurrently.

Significant performance improvements can be obtained by maintaining less state information. Less time is spent maintaining the state, and computations can be reordered without concern for maintaining state information. Consequently, heuristics may be used to select more optimal orderings. Finally, it should be noted that the dramatically increased computational ability of *DADO* can be exploited to rapidly compute matches when required—the importance of saving the prior matches is thereby reduced.

LOGIC PROGRAMMING

Logic programming is a formalism based upon symbolic logic. PROLOG is probably the best known example. According to this formalism, a clause consists of logical first-order terms that must be satisfied. Frequently, the clauses may be viewed as a sequence of conjunctions and disjunctions, and thus an AND/OR tree is commonly used to represent the programs. Logic programming allows a problem to be specified as general rules and data structures that will be expanded to solve a directive.

Logic programming has attracted a great deal of attention as a medium for the development of software for parallel execution. Two major factors contributing to this perception are the demonstrated suitability of logic programming for the expression of a wide variety of software tasks, and the identification of several sources of parallelism inherent in the logic formalism itself. Thus logic programming languages appear to offer a framework in which programs naturally lend themselves to efficient parallel execution, but in which the programmer need not be overly cognizant of this goal.

A Logic Programming Language (LPS)[16,15,18] is under development at Columbia. The key aspects of LPS are *unification* and *reconciliation.* Unification finds a substitution to transform two terms into identical terms. There is a most general unifier such that all other unifiers are instances of it. The unification processes can be performed in many processors, each one working on a particular goal.

Reconciliation is a process that determines if two substitutions are compatible, and produces the most general substitution if one exists. It is an essential part of the LPS algorithms, and makes the various unifiers consistent. In this manner the independent unifications give rise to one answer.

We may view the proof search as a perusal of goal lists. The LPS search tree, in comparison to the standard sequential algorithm used by Prolog, is characterized by

1. Shorter paths to leaves
2. Earlier termination of unproductive paths
3. Earlier consideration of most goals, causing earlier branching but not necessarily higher branching factors
4. A substantially reorganized leaf structure, resulting in a different order to the construction of solutions.

The unification phase of the LPS algorithm places different facts into each PE, and then transmits a goal list from the CP into the PE network. Each PE unifies with many goals, producing unifiers that are tagged with a *level number* to identify the goal whose unification gave rise to the binding set.

The second phase is the reconciliation phase. It is also known as a "join phase" due to similarity with the operation of an equi-join over a set of relational database operations. A heuristic for ordering the join phase can, in most circumstances, keep join phase communication close to minimal. The join phase is coordinated by the CP, to allow communication of binding sets around the network.

The last task to be performed upon discovery of a successful proof is the composition of the various substitutions that were generated along the way. In the LPS implementation, the substitution phase is accomplished by transmitting prior reconciliation history to the PE network, and computing in each PE the composition of that substitution with any new reconciliation.

These algorithms have been implemented (on a sequential computer using PSL) in order to uncover problems in parallel execution of logic programs. Planning for an implementation of an LPS interpreter to be executed on the prototype *DADO2* machine is presently underway.

ARTIFICIAL INTELLIGENCE: DTEX GENERIC EXPERT SYSTEM TOOL

DTEX is a decision tree based expert system building tool. DTEX organizes knowledge into a decision graph, which is appropriate for problems that can be expressed as a hierarchy of relationships. It uses nodes to represent properties, and edges to represent the values corresponding to properties of the source node. This design represents conjunctions of disjunctions as appropriately structured graphs. Explicit AND/OR edges are not needed because the system forward-chains towards its decision.

DTEX[22] consists of three main modules: strategy, explanation, and knowledge acquisition. The strategy phase is divided into *ask, decide,* and *next* phases whereby the system can collect information, select an appropriate edge, and continue in the next phase.

Explanation provides a recapitulation of the inference processes. In addition, it responds by supplying the rationale behind a link as a useful technique to convince or teach the user.

The system adds new knowledge through interaction between DTEX and the specialist. The knowledge acquisition phase will perform extensive consistency checking in addition to syntactic checks (such as "*if* valid values are integers from 0 to 5, *but* the new node could only be reached if values are between 3 and 5, *and* the purpose is to consider the new node, *then* only allow edges with values between 3 and 5"). Semantic checking is performed by building frames with appropriate semantic information. This requires the expert to express meanings carefully and thereby prevents mistakes. A preliminary version of DTEX has been developed in the domain of recommending treatment for adolescent idiopathic scoliosis.

SUMMARY

A prototype machine, named *DADO1,* has been operational at Columbia University since April 25, 1983. As noted, *DADO1* consists of 15 Intel 8751 microprocessors. These 15 PEs, interconnected in a complete binary tree, serve as a testbed for the development of software for a larger prototype currently under construction. *DADO2* consists of 1023 Intel 8751-based PEs, which will provide in excess of 570 MIPS, and is expected to be completed in the coming months.

DADO2 is not viewed as a performance machine, but rather as a laboratory vehicle to investigate fine-grain processors. We expect *DADO2* to achieve significant performance improvements of AI software;[28, 20] but more important, it will provide a testbed for the next generation machine, which is expected to achieve the stated goals of DARPA's Strategic Computing Program for symbolic multiprocessors. Although the binary tree topology may cause severe communication bottlenecks for some parallel operations, we expect that *DADO* will achieve very impressive performance for applications where computation dominates over communication. We have found this to be the case for AI production systems, for example. Since tree structures offer many favorable advantages for hardware implementation, *DADO* will offer very attractive cost performance for many of these important applications.

ACKNOWLEDGMENTS

This research was conducted as part of the *DADO* project. It was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0165, and the New York State Science and Technology Foundation NYSSTFCAT(84)-15, as well as by grants from AT&T, DEC, Hewlett Packard, IBM, Intel, and Valid Logic Systems.

The work was done by the following members of the *DADO* project: Michael van Biema, Jens Christensen, Andrew Comas, Eugene Dong, Beverly Dyer, Douglas Gordin, Mark D. Lerner, Andy Lowry, Janet Lustgarten, Gerald Q. Maguire Jr., Chris Maio, Russell Mills, Daniel P. Miranker, Alexander Pasik, Richard Reed, Pandora Setian, Salvatore Stolfo, Steve Taylor, Nancy Yee, Philip Yuen.

REFERENCES

1. Stolfo S. J., and D. E. Shaw. *Specialized Hardware for Production Systems.* Department of Computer Science, Columbia University, August, 1981.
2. Stolfo, S. J. "Knowledge Engineering: Theory and Practice." *Proceedings of IEEE Trends and Applications,* Gaithersburg, MD, 1983.
3. Stolfo, S. J., D. P. Miranker, and D. E. Shaw. "Architecture and Applications of DADO: A Large Scale Parallel Computer for Artificial Intelligence." *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany: IJCAI, August 1983.
4. Stolfo, S. J. *On the Design of Parallel Production System Machines: What's in a LIP?* Department of Computer Science, Columbia University, 1984.
5. Stolfo, S. J., and D. E. Shaw. "DADO: A Tree-Structured Machine Architecture for Production Systems." *Proceedings of AAAI-82.* Pittsburgh: Carnegie-Mellon University, 1982.
6. Stolfo, S. J. *The DADO Parallel Computer.* Technical report, Department of Computer Science, Columbia University, August, 1983.
7. Stolfo, S. J., and D. P. Miranker. "DADO: A Parallel Processor for Expert

Systems." *Proceedings of the 1984 International Parallel Processing Conference,* Michigan, IEEE Computer Society Press, 1984.

8. Shaw, D. E., S. J. Stolfo, H. Ibrahim, B. Hillyer, G. Wiederhold, and J. A. Andrews. "The NON-VON Database Machine: A Brief Overview." *Database Engineering 4,* 2 (December 1981).
9. Browning, S. "Hierarchically Organized Machines." In C. Mead and L. Conway (eds.), *Introduction to VLSI Systems.* Reading, Mass.: Addison-Wesley, 1978.
10. Flynn, M. J. "Some Computer Organizations and Their Effectiveness." *IEEE Transactions on Computers,* Vol. 25 (September 1972).
11. Miranker, D. P. *Performance Analysis of DADO by Queueing Network Analogy.* Department of Computer Science, Columbia University, November 1983.
12. Taylor, S., C. Maio, S. J. Stolfo, and D. E. Shaw. *PROLOG on the DADO Machine: A Parallel System for High Speed Logic Programming.* Department of Computer Science, Columbia University, 1983.
13. Stolfo, S. J., D. Miranker, and M. Lerner. *PPL/M: The Systems Level Language for Programming the DADO Machine.* Department of Computer Science, Columbia University, 1984.
14. Taylor, S., D. Tzoar, and S. J. Stolfo. *Unification in a Parallel Environment.* Department of Computer Science, Columbia University, 1984.
15. Taylor, S., A. Lowry, G. Maguire, and S. J. Stolfo. "Logic Programming Using Parallel Associative Operations." *Proceedings of the International Logic Programming Symposium,* Atlantic City, N.J., IEEE Computer Society Press, February, 1984.
16. Taylor, S. *LPS, A Logic Programming System: Motivations and Goals.* Department of Computer Science, Columbia University, 1984. (In preparation)
17. van Biema, M., M. D. Lerner, G. Q. Maguire, and S. J. Stolfo. *PSL: A Parallel LISP for the DADO Machine.* Department of Computer Science, Columbia University, February, 1984.
18. Lowry, A., S. Taylor, and S. J. Stolfo. "LPS Algorithms: A Detailed Examination and Critical Analysis." *Proceedings of the International Conference on Fifth Generation Computer Systems.* Tokyo: Proc. Institute for New Generation Computer Technology, November 1984.
19. Lerner, M., M. van Biema, and G. Q. Maguire Jr. *A LISP Compiler for the DADO Parallel Computer.* Department of Computer Science, Columbia University, November 1984.
20. Gupta, A. *Implementing OPS5 Production Systems on DADO.* Department of Computer Science, Carnegie-Mellon University, 1984.
21. Miranker, D. *HerbAl; A Production System for the DADO Machine.* Department of Computer Science, Columbia University, 1984. (In preparation)
22. Pasik A., D. Gordin, L. Finkel, and J. Lustgarten. "XNET/AIS: An Expert System that Recommends Treatment for Adolescent Idiopathic Scoliosis." Unpublished course project, Columbia University, Department of Computer Science, 1984.
23. Intel Corporation. *PL/M-51 User's Guide for the 8051 Based Development System.* Intel, 1982. Order Number 121966.
24. Griss M. L., and A. C. Hearn. "A Portable LISP compiler." *Software—Practice and Experience,* 11 (June 1981), pp. 541–605.
25. Griss, M. L., E. Benson, and G. Q. Maguire Jr. "PSL: A Portable LISP System." *Symposium on LISP and Functional Programming,* ACM, Pittsburgh, August 1982, pp. 88–97.
26. Newell, A. "Production Systems: Models of Control Structures." In W. Chase (ed.), *Visual Information Processing.* New York: Academic Press, 1973.
27. Rychener, M. *Production Systems as a Programming Language for Artificial Intelligence.* Ph.D. thesis, Carnegie-Mellon University, Department of Computer Science, 1976.
28. Stolfo, S. J. *A Note on Implementing OPS5 Production Systems on DADO.* Department of Computer Science, Columbia University, June 1984.
29. Pasik A., and M. Schorr. "Table-driven Rules in Expert Systems." *SIGART Newsletter,* 87 (January 1984), pp. 31–33.
30. Vesonder, G. T., S. J. Stolfo, J. Zalinski, F. Miller, and D. Copp. "ACE: An Expert System for Telephone Cable Maintenance." *Proceedings of the International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany: IJCAI, August 1983.
31. Vesonder, G. T., S. J. Stolfo, J. Zalinski, F. Miller, and J. Wright. "The Application of Artificial Intelligence Technology for Managing the Local Telephone Network." *Proceedings of the International Conference on Computer Communication,* Australia, November 1984.
32. Forgy, C. L. *OPS5 User's Manual.* Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, July 1981.
33. Miranker, D. P. "Performance Estimates for the DADO Machine: A Comparison of TREAT and RETE." *Proceedings of the International Conference on Fifth Generation Computer Systems.* Tokyo: Proc. Institute for New Generation Computer Technology, November 1984.

Dense matrix operations on a torus and a boolean cube

by S. LENNART JOHNSSON
Yale University
New Haven, Connecticut

ABSTRACT

Algorithms for matrix multiplication and for Gauss-Jordan and Gaussian elimination on dense matrices on a torus and a boolean cube are presented and analyzed with respect to communication and arithmetic complexity. The number of elements of the matrices is assumed to be larger than the number of nodes in the processing system. The algorithms for matrix multiplication, triangulation, and forward elimination have 100% processor utilization, except for a latency period proportional to the diameter of the system. The constant of proportionality is small. Distributed one-to-all routing algorithms that guarantee completeness and uniqueness, and terminate after k steps for a k-cube are also given.

INTRODUCTION

Matrix computations and the solution of linear systems of equations have been extensively studied in the past. Most algorithms for matrix multiplication and for the solution of dense systems of equations have a sequential arithmetic complexity of $O(n^3)$. Algorithms of a lower asymptotic arithmetic sequential complexity are known,[1] but have a more complex control structure. Compatibility between machine architecture and data and control structures is of great importance for the efficient implementation of most computations. The efforts expended in developing efficient algorithms for vector machines, and vectorizing compilers, provide ample evidence thereof.

The match between data and control structures and the machine architecture is even more critical for efficiency in a highly concurrent system with limited communication capability. Systolic algorithms are devised with particular attention to data movement in systems of fine grain. Most such algorithms for matrix operations are devised from the traditional algorithms of order $O(n^3)$ complexity. The fact that the operations in rows and columns are only partially ordered is exploited for concurrency. The algorithms vary in the way the inner products are computed over time and space. A brief survey is contained in Johnsson.[2] A methodology for synthesizing systolic algorithms is presented by M. C. Chen.[3] A common characteristic of many systolic algorithms is that the initial data and the result, or both, are stored outside an array having a topology that perfectly matches the communication needs of the algorithm. The issue of storage management is generally ignored.

The algorithms described here are devised for multiprocessor systems consisting of processors with sufficient local storage to allow the initial data and the result to be stored within the multiprocessor. The granularity of operations and the size of the processors with their local storage, referred to as nodes, are in general larger than envisioned in systolic arrays. Although it is possible to design processors of most sizes with a wide range of communication bandwidths relative to arithmetic bandwidths, the larger a processor gets, the lower the communication tends to become.[4,5] We refer to multiprocessor systems with a large number of nodes, each with substantial local storage, as *ensemble architectures*. Furthermore, we assume that these architectures are of the MIMD type.[6] Familiar systolic algorithms[7] can be modified to reduced ensemble sizes by folding the computations in space,[8] and by space-time transformations to yield algorithms with reduced data movement. The folding operations result in additional computations being performed in a node with opportunities for optimizing the granularity of operations with respect to architectural parameters. The emphasis here is on algorithms with reduced data movement.

We assume that the processors with their local storage are interconnected either as a torus, i.e., a mesh with end-around connections in both dimensions, or as a boolean cube. Our algorithms for matrix multiplication and solution of dense systems of equations are derived from algorithms of sequential complexity $O(n^3)$. The focus is on data and control structures for efficient communication in treating large problems on small machines. The system of equations is written $AX = B$, with A being an N-by-N matrix and B an N-by-R. The number of nodes in the ensemble is $K \leq N^2$. For the torus, we assume that there are $\sqrt{K}$ nodes in each dimension, and for the cube, that $K = 2^{2k}$. A matrix can be embedded in a boolean cube of even dimension by assigning half of the address space to row indices and half to column indices. Embedding by binary encoding of the indices and by encoding the indices in a *binary-reflected* Gray code are considered. The Gray code embedding preserves proximity, whereas the binary encoding does not. The time for communicating one floating-point number is t_c, the time for division is t_d, and t_a is the time for multiplication, addition, or subtraction. For the complexity estimates, we assume that communication and computation can take place concurrently, and that communication between a pair of processors can occur in either direction, also concurrently.

We first describe a few matrix multiplication algorithms for ensembles configured as meshes with end-around connections. The algorithms differ in the way data is communicated, and in the control structure of the algorithms. One of the algorithms is similar to one by Cannon,[9] but the rate at which successive matrix multiplications can be pipelined is higher in our algorithm if the communication bandwidth is less than the arithmetic bandwidth. The increased rate of pipelining is in part obtained by assuming that the ensembles are of the MIMD type. One of the algorithms has minimal communication under the assumption that inner products are accumulated *in-place*. The complexity of the new algorithms for the torus is of the form $\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil \lceil R/\sqrt{K}\rceil\sqrt{K}\max(2t_a,t_c) + \alpha\sqrt{K} + \mathit{const}$ for the operation $C \leftarrow A \times B + D$ where A is N by M and B is M by R. α varies between 2 and 4. The speed-up of the dominating term is linear in the number of processing elements.

The algorithms for the torus are modified to take advantage of the increased communication capabilities of a boolean cube configured ensemble. Except for a time accounted for in the lower order term, the algorithms devised for the torus fully use the processors. This term is reduced further in order to become of an order $O(\log_2 K)$ for three of the cube algorithms, but only by a constant factor for one of the multiplication algorithms (one that has a data movement compatible with the presented Gaussian elimination algorithm).

For the boolean cube, two distributed one-to-all routing algorithms are given. We prove completeness and uniqueness upon termination for these algorithms. For one of the algorithms, it is shown that with source nodes ordered by the integers their addresses encode, the order of message arrival for all nodes is the same as the order of distribution.

For the solution of dense systems of equations, we consider Gauss-Jordan and Gaussian elimination. The data movement in the elimination algorithms is similar to one of the matrix multiplication algorithms, and the speed-up of the dominating term is linear. For the backsubstitution, we discuss two approaches, one feasible for a large number of right hand sides, and one for few right hand sides. The latter yields linear speed-up for a large range of ensemble sizes at the expense of increased temporary storage as compared to the former, which has linear speed-up for a large number of right hand sides.

MATRIX MULTIPLICATION

Multiplication on a Torus

The product matrix is denoted by $C(C = A \times B + D)$, and for the algorithms presented here, A, B, C, and D are assumed to be stored in row major order. Furthermore, all of the algorithms considered accumulate inner products *in-place*. The communication and control structures are first defined for the multiplication of two $\sqrt{K}$ by $\sqrt{K}$ matrices on a $\sqrt{K}$ by $\sqrt{K}$ torus, and are then generalized to arbitrarily large matrices with A being an N by M matrix, and B an M by R matrix where $N,M,R \geq \sqrt{K}$.

The matrices are assumed to be stored *cyclicly*, i.e., processor row i stores equations $\{k|i = k \bmod \sqrt{K}, 0 \leq k \leq N-1\}$, and processor column j stores columns $\{l|j = l \bmod \sqrt{K}, 0 \leq l \leq M\}$ of an N by M matrix. The matrices are partitioned into $\sqrt{K}$ by $\sqrt{K}$ submatrices by way of storage. The quotient in dividing the row index by $\sqrt{K}$ is the same for all elements of a submatrix. The same property is true for the column index. Another natural storage scheme is *consecutive* storage, in which submatrices of size $N/\sqrt{K}$ by $M/\sqrt{K}$ for a N by M matrix are stored in each node. More precisely, node (p,q) stores matrix elements (i,j) such that $p = \lfloor i\sqrt{K}/N]$, and $q = \lfloor j\sqrt{K}/M \rfloor$.

The apparent granularity of operations for consecutive storage is greater than for the cyclic storage scheme. However, in the fine grain of operations suggested by the cyclic storage scheme, it may be desirable to combine operations to reduce the influence of overhead in communication and arithmetic operations. Conversely, it may be desirable to reduce the granularity suggested by the consecutive storage scheme by partitioning the block operations in order to increase the concurrency. The optimization yields the same result for matrix multiplication, regardless of which of the two storage schemes and associated algorithms is used as origin. For Gaussian elimination, the consecutive storage scheme yields lower processor utilization in that after the first $N/\sqrt{K}$ elimination steps, one row and column of processors become idle in Gaussian elimination, and one column in Gauss-Jordan elimination; after another $N/\sqrt{K}$ elimination steps, yet another row and column of processors become idle in Gaussian elimination, etc.

We assume the cyclic storage scheme below, because it tends to offer a better insight into the data and control structures of the algorithms.

Multiplication by rotation, MMT1

Clearly, for A and B stored in row-major order and of dimension $\sqrt{K}$ by $\sqrt{K}$, the multiplication requires a time proportional to $\sqrt{K}$, if the concurrent algorithm is based on the traditional inner product algorithm for matrix multiplication. The number of arithmetic operations is $K(2\sqrt{K}-1)$, and the number of nodes is K. Cannon describes an algorithm for SIMD-type ensembles configured as toruses.[9] The algorithm has two phases, a set-up phase in which the matrices are aligned such that all nodes in the two-dimensional array can perform a multiply-add operation, and a multiplication phase in which both matrices are repeatedly shifted one step followed by a multiply-add operation.

In the set-up phase, row i of A is shifted i steps in the direction of decreasing column indices, and column j of B is shifted j steps in the direction of decreasing row indices. Each processor use three registers denoted by E, F, and G. Initially, $E(i,j) \leftarrow A(i,j)$, $F(i,j) \leftarrow B(i,j)$, and $G(i,j) \leftarrow D(i,j)$ where $(i,j) = \{0,1,\ldots,\sqrt{K}-1\} \times \{0,1,\ldots,\sqrt{K}-1\}$. The set-up phase consists in the operations $E(i,j) \leftarrow E(i,(j+i) \bmod \sqrt{K})$, $F(i,j) \leftarrow F((i+j) \bmod \sqrt{K}, j)$.

Clearly, $E(i,j) \times F(i,j)$ are valid product terms for $(i,j) = \{0,1,\ldots,\sqrt{K}-1\} \times \{0,1,\ldots,\sqrt{K}-1\}$. The set-up phase implements the skewing of data streams required for synchronization, as seen in many systolic algorithms.[7,10] In the multiplication phase, the following computations are performed in Cannon's algorithm.

```
for k = 1 step 1 until $\sqrt{K}$ do
    for all (i, j) in parallel do
        $G(i,j) \leftarrow G(i,j) + E(i,j) \times F(i,j)$
        $E(i,j) \leftarrow E(i,(j+1) \bmod \sqrt{K})$
        $F(i,j) \leftarrow F((i+1) \bmod \sqrt{K}, j)$
    end
end
```

The complexity of this algorithm is $2(\sqrt{K}-1)t_c + 2\sqrt{K}t_a$. Several variations of this algorithm exist. For instance, in an ensemble of the MIMD type, it is not necessary to first explicitly carry out the set-up phase and then proceed to the multiplication phase. The alignment and multiplication phases can be overlapped such that at step j, $j < \sqrt{K}$, processors $(i,j) = \{0,1,\ldots,j\} \times \{0,1,\ldots,j\}$ perform multiply-add operations. At step j, row k of A, $k \leq j$ has been shifted cyclicly $j-k$ steps. Similarly, column l of B, $l \leq j$, has been shifted $j-l$ steps. For $\sqrt{K} \leq j < 2\sqrt{K}$, rows and columns continue to be shifted cyclicly, but only processors $(i,j) = \{j \bmod \sqrt{K}+1,\ldots,\sqrt{K}-1\} \times \{j \bmod \sqrt{K}+1,\ldots,\sqrt{K}-1\}$

perform multiply-add operations; however, the complexity is the same as that of Cannon's algorithm.

One disadvantage of Cannon's algorithm is that one row and one column are rotated $2\sqrt{K}-1$ steps, one row and column $2\sqrt{K}-2$ steps, etc. With unidirectional communication and inner products accumulated in-place, one revolution of any matrix element suffices. The additional communication in Cannon's algorithm is because of the required synchronization between row and column operations, and the SIMD-mode of operation. If the communication bandwidth is at least as high as the arithmetic bandwidth, i.e., $t_c \leq t_a$, and if communications and arithmetic operations take place concurrently, multiple matrix multiplications can be pipelined such that the arithmetic operations are limiting the performance. However, if $t_c > t_a$, it is of interest to find algorithms with reduced communication requirements.

We next describe an algorithm that uses communication in both directions on every link, to improve the rate at which multiple matrix multiplications can be pipelined. We then present an algorithm in which each element is routed only $\sqrt{K}-1$ steps, the minimum possible with inner products accumulated in place. Both algorithms are based on the outer product formulation of matrix multiplication, and coincide if the communication delay is zero. However, their communication and control structure differs, thereby resulting in different latency for nonzero communication delays. Multiple matrix multiplications can be pipelined at the same rate for both algorithms (in which it is assumed that that the ensemble is of the MIMD type).

Multiplication by reflection in processor row and column 0, MMT2

In this algorithm, the computation of all outer products is initiated in processor $(0,0)$, and propagates towards processor $(\sqrt{K}-1, \sqrt{K}-1)$. End-around connections are not used. The computational *wave fronts,* i.e., the set of processors that at a given time are in the same state of computation, lie on antidiagonals. The elements of each row of the matrix A are sent along the respective row towards the first column, thereby preserving the order of elements within each row. Correspondingly, the elements of B are sent in the direction of decreased row index, thus preserving the order of elements within columns. When an element of A reaches the first column, it is sent back along the same row. Similarly, the elements of B are reflected in row 0. On reflection, and on their path towards the last column (A) and row (B), respectively, the elements of A and B are multiplied and added to the local partial product sum. The computations of successive rows and columns are delayed with respect to each other for synchronization. Processor $(0,0)$ receives element a_{01} from processor $(0,1)$, and element b_{10} from processor $(1,0)$, and performs a multiply-add operation on elements a_{00}, b_{00}, and d_{00} during the first cycle. Processor $(\sqrt{K}-1, \sqrt{K}-1)$ sends element $a_{\sqrt{K}-1,\sqrt{K}-1}$ to processor $(\sqrt{K}-1, \sqrt{K}-2)$, and element $b_{\sqrt{K}-1,\sqrt{K}-1}$ to processor $(\sqrt{K}-2, \sqrt{K}-1)$ during cycle $\sqrt{K}-2$, and starts receiving elements during cycle $2\sqrt{K}-3$. Processor $(\sqrt{K}-1, \sqrt{K}-1)$ performs multiply-add operations during cycles $2\sqrt{K}-2$ through $3\sqrt{K}-3$. The last active cycle for processor $(0,0)$ is $\sqrt{K}-1$. It is straightforward to derive the following complexity estimate.

Proposition 1. Performing the operation $C \leftarrow A \times B + D$ by the outer product algorithm on a mesh by reflections of the elements of A in the first column and the elements of B in the first row requires a time of at most $(\sqrt{K}-1)\max(2t_a, t_c) + 2(\sqrt{K}-1)t_c + 2t_a$, with the matrices and the mesh being of size $\sqrt{K}$ by $\sqrt{K}$, assuming that communication and arithmetic operations can be performed concurrently.

The factor of 2 multiplying the term $(\sqrt{K}-1)$ can be reduced to 1 by starting the computations from both processor $(0,0)$ and $(\sqrt{K}-1, \sqrt{K}-1)$. Multiple matrix multiplications can be pipelined at the rate of $\sqrt{K}\max(2t_a, t_c)$. An N by M matrix A is partitioned into $\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil$ blocks and an M by R matrix B is partitioned into $\lceil M/\sqrt{K}\rceil\lceil R/\sqrt{K}\rceil$ blocks, each of size $\sqrt{K}$ by $\sqrt{K}$.

Proposition 2. The operation $C \leftarrow A \times B + D$ with A being an N by M matrix, B an M by R, and C and D being N by R matrices, respectively, requires a time of $(\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil \lceil R/\sqrt{K}\rceil\sqrt{K} - 1)\max(2t_a, t_c) + (\sqrt{K} - 1)t_c + 2t_a$ when performed by the outer product algorithm on a mesh by reflections of A in column 0 and of B in row 0.

Multiplication by initiating outer products on the diagonal, MMT3

The algorithm described here routes a matrix element only $\sqrt{K}-1$ steps. The computation of the outer product of column k of A and row k of B is initiated in processor (k, k). With unidirectional communication, elements of A are sent in the direction of increasing column index, and elements of B in the direction of increasing row index. If the kth outer product is initiated at time t_k, then processor $((k+i)\bmod\sqrt{K}, k)$ sends element $a_{(k+i)\bmod\sqrt{K},k}$ to processor $((k+i)\bmod\sqrt{K}, k+1)$ at time $t_k + it_c$, $i = \{1, 2, \ldots, \sqrt{K}-1\}$; processor $((k+i)\bmod\sqrt{K}, k+1)$ forwards the element to processor $((k+i)\bmod\sqrt{K}, k+2)$ during the following cycle, etc. A similar relation holds for the elements of the columns. A processor performs the multiply-add operation upon receipt of one element from each of A and B. A new outer product can be initiated every $2t_c + \max(2t_a, t_c)$ units of time.

Proposition 3. The computation of the product of two $\sqrt{K}$ by $\sqrt{K}$ matrices on a torus of matching size by the outer product method (initiating successive outer products on the diagonal), and with unidirectional communication, requires a time of $(\sqrt{K}-1)\max(2t_a, t_c) + 4(\sqrt{K}-1)t_c + 2t_a$.

The complexity of this algorithm is higher than the complexity of algorithm MMT2, due in part to the requirement that communication be unidirectional. The complexity can be reduced by $\sqrt{K}t_c$ if two-way communication is allowed. The wavefronts for two-way communication form diamonds.

Multiple matrix multiplications can be pipelined at the rate of one new product every $\max(2t_a, t_c)$ units of time. Each processor receives and sends $\sqrt{K}-1$ elements of A and B, and performs $\sqrt{K}$ multiply-add operations. Figure 1 shows

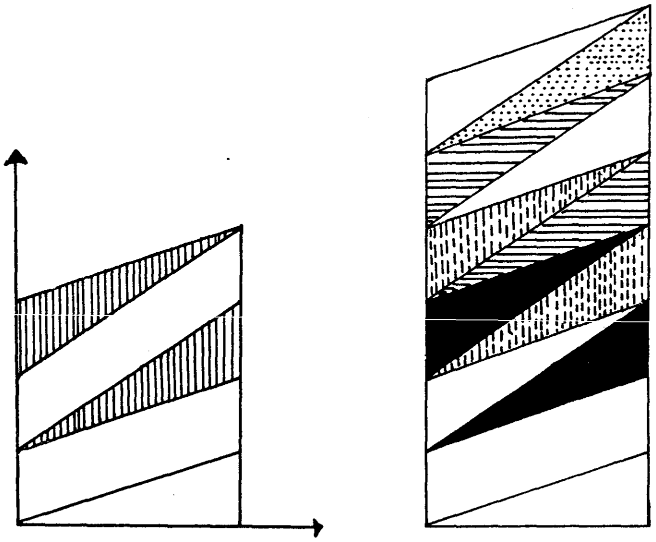


Figure 1—Stylized timing diagrams for matrix multiplication on a torus

the periods of activity of row 0 and the pipelining of successive matrix multiplications in a stylized fashion. Proposition 4 follows.

Proposition 4. Multiplication of an M by R matrix B by an N by M matrix A using an outer product algorithm with initiation of products on the diagonal of a $\sqrt{K}$ by $\sqrt{K}$ torus can be accomplished in time $(\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil\lceil R/\sqrt{K}\rceil - 1)\sqrt{K}\max(2t_a, t_c) + 2(\sqrt{K} - 1)t_c + 2t_a$, with A, B, C and D, $C = A \times B + D$, stored in row-major order.

Multiplication on a Boolean Cube

The diameter of a boolean cube of $K = 2^{2k}$ processors is $2k$. This fact can be taken advantage of, and the latency of the multiplication algorithm reduced from order $\sqrt{K}$ on the torus to order $\log_2 K$ on the cube. We define the latency as the contribution to the complexity caused by global communication in the case of nonnegligible communication time (or *retiming,*[11] in the terminology used for systolic computations). No further reduction of the complexity is possible, since the processors in our algorithms are fully utilized with the exception of the latency period. Dekel et al. describe an algorithm with latency $1/2\log_2 K$ for matrices embedded in a boolean cube by a binary encoding of row and column indices, each in half of the address space, i.e., matrix element (i, j) is embedded in the processor with address $(r_{k-1} \ldots r_0 s_{k-1} \ldots s_0)$, where $r_{k-1} \ldots r_0$ is the binary encoding of i and $s_{k-1} \ldots s_0$ is the binary encoding of j.[12]

An embedding by an encoding of row and column indices in a *binary-reflected* Gray code[13] preserves proximity of row and column indices. If the rotation performed in the set-up phase in Cannon's algorithm can be performed in a time proportional to k instead of K, then the multiplication phase of the algorithm by Cannon is directly applicable, and the complexity of the algorithm is the same as that of the algorithm by Dekel et al. Furthermore, if the routing in the set-up phase is such that routing paths of elements with distinct origins and destinations intersect at nodes only, then the set-up phase for different matrix multiplications can be pipelined such that the latency only appears once in the complexity, assuming that a processor can support concurrent communication on all its ports. In algorithm MMT2, the purpose of the routing of a matrix element upon reflection is to distribute it to all other nodes in the row or column, respectively. If this one-to-all distribution operation can be accomplished without communication conflicts with the routing of elements to row and column 0, then a modification of algorithm MMT2 to a boolean cube also yields the same complexity. We will prove that there exist routing schemes such that no communication conflicts need occur. In algorithm MMT3 the sole purpose of the communication is to distribute a matrix element to all other nodes in the same row or column. For the adaptation of MMT3 to a cube, we present a one-to-all distribution algorithm alternative to the one used in modifying MMT2.

Multiplication of matrices embedded by binary encoding, MMC1

In the algorithm by Dekel et al.,[12] the set-up phase consists in the rearrangement $E(i,j) \leftarrow E(i, i \oplus j)$, and $F(i,j) \leftarrow F(i \oplus j, j)$. Clearly, $E(i,j) \times F(i,j)$ are valid product terms for all i and j. With i and j encoded in separate subcubes, the data movement in the set-up phase is confined to these subcubes. Consider the rearrangement of the matrix A. The routing required in the subcube defined by i is $j \leftarrow i \oplus j$. This routing consists of exchanges in different dimensions, namely those in which the binary encoding of i has a bit equal to 1. Each dimension is routed only once. Hence, routing the dimensions in the same order for all j guarantees that any interconnection is only traversed by one element in a given direction.

Lemma 1. The set-up phases for multiple matrix multiplications can be pipelined such that the total time for the set-up phase for P independent matrix products is $(P + k - 1)t_c$, given that a processor can support concurrent communication on all of its ports by routing the dimensions traversed by the elements in the same order for all elements.

In the multiplication phase, exchanges of elements are performed in the different dimensions in the order specified by the transition sequence in a binary reflected Gray code.[12]

Multiplication by rotation of Gray code encoded matrices, MMC2

With the matrix embedded according to a binary-reflected Gray code encoding, and with the above algorithm for matrices embedded by binary encoding of row and column indices, one approach to the multiplication problem is to convert the embedding to an embedding according to a binary encoding. A k-bit Gray code encoding can be converted to a binary encoding in $k - 1$ routing steps. The highest order bit in the binary and Gray code encodings coincide. The second highest order bits differ in the Gray code and binary encodings of the upper half of the indices. An exchange operation between processors differing in the second highest order address bit, and where the highest order address bit is 1, guarantees that the matrix elements are in the correct subcubes of dimensionality $k-2$. By the construction of the binary-reflected Gray code, the process can be repeated recursively.[5] The exchange operation takes place on an increasing number of subcubes of reduced dimensionality. Each dimension is routed only once.

Lemma 2. An embedding according to a Gray code encoding can be converted to an embedding according to a binary encoding in $k-1$ routing steps such that the path of elements with distinct origins and destinations share no edges.

For P matrix multiplications, the total time for conversion of embeddings and initial alignment is at most $(5(P+k)-7)t_c$. At most $2(k-1)+2P-1$ communications are needed for each of the conversions of the embeddings.

If the routing needed for the rotations in Cannon's algorithm can be performed with properties similar to the routing in the set-up phase in the algorithm by Dekel et al., then there is no need for conversion of embeddings. The maximum number of routing steps for a rotation is clearly k. In order that the set-up phase be completed in time k, it is required that the routing be such that no two elements compete for the same communication link. Furthermore, for a maximum degree of pipelineability, it is necessary that a communications link be used only once in a given direction.

I have proved that any routing $i \rightarrow i+2^r$, $r>0$ requires precisely two routing steps.[14] Indeed, one of the routed dimensions is dimension $r-1$. The other dimension is one of r through k, and depends on i and r. Any rotation can be performed as a sequence of rotations of the form 2^r, where the set of r values is determined by the binary encoding of the size of the rotation. The set-up phase requires a time of at most $2k$. We claim without proof that the set-up phase for P matrix multiplications requires a time of at most $2(P+k-1)t_c$, and Proposition 5 follows.

Proposition 5. The time for the operation $C \leftarrow A \times B + D$, with A, B, C, and D embedded by separately encoding row and column indices in a binary reflected Gray code, and of dimensions N by M, M by R, and N by R, respectively, can be performed in a time of $\sqrt{K}\lceil N/\sqrt{K}\rceil\lceil R/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil\max(2t_a, t_c) + 2((\lceil N/\sqrt{K}\rceil + \lceil R/\sqrt{K}\rceil)\lceil M/\sqrt{K}\rceil + k - 1)t_c$ on a $2k$-cube using multiplication by rotation.

Multiplication by reflection of Gray code encoded matrices, MMC3

This algorithm is a modification of algorithm MMT2 to a boolean cube configured ensemble. We modify the distribution of matrix elements within a row or a column upon reflection. All reflected elements originate from either row 0 or column 0. Assume that the same routing algorithm is applied to rows and columns, and that the routing distance of column j from column 0 is $r(j)$. Similarly, the routing distance of row i from row 0 is $r(i)$. Then, multiplication in node $(0,j)$ starts at time $r(j)$, in node $(i, 0)$ at time $r(i)$, and in node (i,j) at time $r(i)+r(j)$, assuming a synchronous mode of operation to simplify the argument. The product $a_{ik}b_{kj}$ is easily seen to take place at time $r(i)+r(j)+k$, for all i, j and k. Synchronization between row and column operations is accomplished by initiating computations at the proper times, and by applying the same routing algorithm in both dimensions. Let a processor address be $(i_{k-1}i_{k-2}\ldots i_0)$, and let the highest order bit that is 1 be r, with $r=-1$ for processor $(00\ldots0)$. One possible one-to-all distribution algorithm is as follows:

> Processor $(00\ldots01i_{r-1}i_{r-2}\ldots i_0)$ receives a value from processor $(00\ldots00i_{r-1}i_{r-2}\ldots i_0)$
>
> Processor $(0\,..\,01i_{r-1}\ldots i_0)$ sends the received value to all processors $(0\,..\,0i_s0\,..\,01i_{r-1}\ldots i_0)$, $i_s=1$, $s=\{r+1,\ldots,k-1\}$

The routing is performed by complementing leading zeroes (CLZ). Required properties of a one-to-all distribution algorithm for matrix multiplication are completeness, i.e., that all nodes have received the message upon termination, and that it indeed terminates. Desirable properties are uniqueness, i.e., that a processor only receives a message once, and that the control of the routing algorithm is distributed. Clearly, the CLZ algorithm has distributed control, and routing is terminated in processors where the highest address bit is 1. Messages are always moved towards a processor with a higher address (by 2^i for some $i \geq 0$). Completeness and uniqueness is proved by induction on the number of dimensions of the cube.

Lemma 3. Routing by the CLZ algorithm guarantees that all nodes receive a message precisely once within at most k routing steps for a k-cube.

For matrix multiplication by reflection, it is also desirable with respect to performance that the routing paths of elements after reflection do not include the same directed communication link as the paths traversed by elements routed towards the point of reflection. With Gray code encoding, the routing of elements towards row or column 0 is always done towards a processor with an address corresponding to the encoding of $j-1$ if the element currently is in a processor with address encoding j.

Lemma 4. The directed routing paths of elements routed towards the point of reflection, and the directed routing paths defined by the CLZ algorithm intersect at nodes only.

The proof is again by induction. The critical observation is that in doubling the size of the cube, the CLZ algorithm uses only the communication links in the added dimension in the $0 \rightarrow 1$ direction. Indeed, the CLZ algorithm uses all those links, and none of the communication links within the cube that has addresses in the upper half of the address space. The routing towards the point of reflection uses only one of the added communication links, and in the $0 \leftarrow 1$ direction.

Proposition 6. The operation $C \leftarrow A \times B + D$, with A, B, C and D stored by separately encoding row and column indices in a binary reflected Gray code, and of dimensions N by M, M by R, and N by R, respectively, using multiplication by reflection in processors embedding matrix column 0 and row 0, can be performed on a boolean $2k$-cube in time $(\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil\lceil R/\sqrt{K}\rceil\sqrt{K}-1)\max(2t_a,t_c)+2kt_c+2t_a$.

Initiating outer products in processors storing diagonal elements of Gray code encoded matrices, MMC4

Algorithm MMT3 can be adapted to a boolean cube. For this algorithm, the origins of the elements subject to distribution are in processors encoding consecutive integers, if the outer products are initiated in order of successively increasing row and column indices. This is not a necessary requirement; however, we make this restriction here to achieve compatibility in the data movement with a Gaussian elimination algorithm without partial pivoting, described later. The one-to-all distribution algorithm described next makes use of a ticket included in the message to inform a processor, upon receipt, which dimensions remain to be routed. The routing order is determined by the source node (but need not be). The routing order is the order in which distinct dimensions are used in the Gray code encoding of the indices $(i+j)\mathrm{mod}(2^k)$ for monotonously increasing values of $j = \{1, 2, 3, \ldots, 2^k - 1\}$. We refer to this one-to-all distribution algorithm as the GIDD algorithm (Gray code Increasing order Distinct Dimensions). Let G_i be the Gray code encoding of i. The algorithm is described in a synchronous mode of operation, to simplify the arguments of the proof of completeness and uniqueness.

One-to-all distribution with node G_i *as source*

Node G_i computes the order in which the cube dimensions shall be routed.

Node G_i sends the message to node $G_{(i+1)\mathrm{mod}(2^k)}$, and marks the appropriate dimension as being routed. Node G_i keeps a copy of the ticket.

Upon receipt of a message and the ticket, a node resends the message with updated ticket in the dimension next in the routing order. A node keeps a copy of the message and the updated ticket, and keeps resending the message in dimensions not yet routed, with the ticket properly updated.

Each node that receives a message can determine the next dimension of communication, or if the distribution is terminated, from the ticket it receives or the copy it keeps. Note that in the mode of operation described above, a node only communicates on one port at a time. The control of the GIDD algorithm is clearly distributed, and termination is assured.

Lemma 5. The GIDD one-to-all distribution algorithm guarantees that all nodes in a k-cube receive a message precisely once within at most k routing steps.

Proof. The proof is by induction. Initially k dimensions need to be routed. In the synchronous mode of operation, one dimension is covered in the first routing step, and two processors have the message with a ticket stating the same $k-1$ dimensions as remaining to be routed. Since all processors have the same order of the dimensions to be routed, it is clear that if after r routing steps, 2^r processors have received the message and have a ticket with the same $k-r$ dimensions remaining to be routed in the same order, one additional routing covers an $r+1$ cube, with all processors having identical tickets. Hence, completeness is guaranteed. Uniqueness is easily established if the source node is node (00. . . . 00). Routing from any other source node is identical after relabeling of the dimensions.

The distribution paths formed by the GIDD algorithm clearly form a spanning tree rooted at the source node. Hence, the completeness, uniqueness, and termination of the GIDD algorithm also holds for an asynchronous GIDD algorithm.

Corollary 1. The GIDD algorithm is order-preserving in the sense that elements distributed by nodes encoding consecutive integers are received in the order distributed. Distributions from nodes encoding consecutive integers can be initiated every three routing steps.

Employing the GIDD routing algorithm for multiplying matrices by the outer product method, and initiating outer products in order of consecutive indices yields the following complexity for multiplication on a $2k$-cube.

Proposition 7. The operation $C \leftarrow A \times B + D$, with A, B, C and D stored by separately encoding row and column indices in a binary reflected Gray code, and of dimensions N by M, M by R, and N by R, respectively, using multiplication by reflection in processors embedding matrix column 0 and row 0, can be performed in time $(\lceil N/\sqrt{K}\rceil\lceil M/\sqrt{K}\rceil\lceil R/\sqrt{K}\rceil \sqrt{K} - 1)\ \max(2t_a, t_c) + (2(\sqrt{K} - 1) + k)t_c + 2t_a$ on a boolean $2k$-cube (assuming that successive matrix multiplications are interleaved).

The complexity of this algorithm is higher than that of algorithm MMC3 by $2(\sqrt{K} - 1)t_c$, the same difference as between MMT2 and MMT3. The initiation point for successive outer products moves along the diagonal. Two communications are necessary to move from one starting point to the following, and with the unidirectional communication assumed in the algorithm, one additional communication to free up the communication link is needed in the next transition.

SOLVING AN N BY N DENSE SYSTEM OF EQUATIONS BY GAUSSIAN ELIMINATION

In this section, we present and analyze with respect to complexity concurrent Gaussian elimination and Gauss-Jordan algorithms for the solution of linear systems of equations on a torus and a boolean cube. The complexity of Gaussian elimination algorithms on a mesh of processors has also recently been investigated by Saad.[15] Algorithms for ring and bus architectures are analyzed by Ipsen, Saad, and Shultz.[16] Elimination type systolic algorithms for mesh configured ensembles are described in other works.[7, 8, 17] The linear system is written $AX = B$, where A is an N by N matrix, and B an N by R. We only consider the case where pivoting on the diagonal yields acceptable numerical behavior. We divide the analysis into several parts, starting with the diagonalization of a matrix by Gauss-Jordan elimination on a torus of matching size, then consider the triangulation of a matrix of arbitrarily large size, and finally, consider the solution of a triangular system of equations.

Gaussian Elimination on a Torus

Diagonalizing a matrix on a torus of matching size

We assume that the matrix A to be diagonalized is stored in row major order in the torus. In diagonalizing A, we use Gauss-Jordan elimination. We express the inverse in factored form, and store the elements of the factors in the nodes of the torus by replacing elements of A when they are no longer needed, as in a storage-efficient sequential algorithm.

$$A^{-1}=J^{N-1}J^{N-2}\dots J^{0},\qquad J^{j}=\begin{array}{cccccc} 1 & & & f_{0j} & & \\ & 1 & & f_{1j} & & \\ & & \ddots & \vdots & & \\ & & & f_{jj} & & \\ & & & & \ddots & \\ & & & f_{N-1j} & & 1 \end{array}$$

We number rows, columns, and factors from 0. The factors are computed in order of increasing index. The elements of a factor are not known until the application of the preceding factor has progressed to a certain stage. In a pipelined system, the elements of a factor become known during successive cycles. The application of a factor can be initiated as soon as an element of it is known, and the application of successive factors can be pipelined. In the application of the jth factor, a multiple of the jth row is added to all other rows. For ease of pipelining the application of successive factors, we choose to communicate pivot rows in the direction of increasing row index, modulo the number of rows. Hence, row j reaches row $(j+k)\bmod\sqrt{K}$ after k communication steps. The data movement is the same as in the outer product matrix multiplication algorithm with initiation of successive outer products on successive diagonal elements (MMT3), except that communicated rows and columns stem from the same matrix. The size of the columns and rows decreases as the computations progress towards the last elimination operation. The following complexity estimate can be derived in a manner similar to the estimate for algorithm MMT3.

Proposition 8. The time for computing the Gauss-Jordan factored form of the inverse for a $\sqrt{K}$ *by* $\sqrt{K}$ matrix on a torus of matching size is at most $(\sqrt{K}-1)(\max(t_c,2t_a)+t_d)+3(\sqrt{K}-1)t_c+t_a$.

The complexity expression in Proposition 8 is lower than the corresponding expression for matrix multiplication by a term of $\sqrt{K}t_c$ due to the fact that the row length in the last elimination operation is 1. The communication time can be reduced if communication in both directions between a pair of processors is allowed. However, the unidirectional communication simplifies pipelining in the case of $N>\sqrt{K}$. The processor utilization as a function of time is shown in Figure 2 for row 0 of the torus.

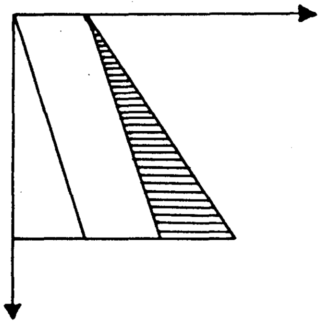


Figure 2—The utilization of processors in row 0 of the torus

Triangulation of an N by N system and forward elimination on R right hand sides

In the algorithm analyzed below, Gauss-Jordan elimination is performed on the pivot block row, and Gaussian elimination on the rows below the block row. The factors $J^0,\dots,J^{N-1}$ are of block form, as is AB. Denote the blocks of J^j containing column j by $J^j_{i\lfloor j/\sqrt{K}\rfloor}$, and the blocks of $j^{j-1}\dots J^0AB$ by $AB^{(j-1)}_{ik}$. The nonzero blocks of $J^j_{i\lfloor j/\sqrt{K}\rfloor}$ are contained in rows $i=\{\lfloor j/\sqrt{K}\rfloor,\dots,\lfloor N/\sqrt{K}\rfloor\}$. The matrix J^j is of the following form.

$$J^{j}=\begin{array}{cccccc} I & & & & & \\ & I & & & & \\ & & \ddots & & & \\ & & & J^{j}_{\lfloor j/\sqrt{K}\rfloor\lfloor j/\sqrt{K}\rfloor} & & \\ & & & \vdots & & \\ & & & J^{j}_{N-1\lfloor j/\sqrt{K}\rfloor} & & I \end{array}$$

The computations in step j are $J^j_{\lfloor j/\sqrt{K}\rfloor\lfloor j/\sqrt{K}\rfloor}\leftarrow AB^{-(j-1)}_{\lfloor j/\sqrt{K}\rfloor\lfloor j/\sqrt{K}\rfloor}$, where the negative superscript denotes the inverse.

$$J^j_{i\lfloor j/\sqrt{K}\rfloor}\leftarrow AB^{(i-1)}_{i\lfloor j/\sqrt{K}\rfloor}AB^{-(j-1)}_{\lfloor j/\sqrt{K}\rfloor\lfloor j/\sqrt{K}\rfloor},$$

$$i=\{\lfloor j/\sqrt{K}\rfloor+1,\dots,\lfloor N/\sqrt{K}\rfloor\},$$

$$AB^{(j)}_{\lfloor j/\sqrt{K}\rfloor k}\leftarrow J^j_{\lfloor j/\sqrt{K}\rfloor\lfloor j/\sqrt{K}\rfloor}AB^{(j-1)}_{\lfloor j/\sqrt{K}\rfloor k},$$

$$k=\{\lfloor j/\sqrt{K}\rfloor+1,\dots,\lceil (N+R)/\sqrt{K}\rceil\}$$

$$AB^{(j)}_{ik}\leftarrow AB^{(j-1)}_{ik}+J^j_{i\lfloor j/\sqrt{K}\rfloor}AB^{(j-1)}_{\lfloor j/\sqrt{K}\rfloor k},$$

$$i=\{\lfloor j/\sqrt{K}\rfloor+1,\dots,\lceil N/\sqrt{K}\rceil\},$$

$$k=\{\lfloor j/\sqrt{K}\rfloor+1,\dots,\lceil (N+R)/\sqrt{K}\rceil\}$$

The matrix $AB^{(j-1)}$ and its storage in the torus is shown in Figure 3. Pivot row j and the jth column of J^j are shaded. Every element (j,l) of the pivot row such that $k=l\bmod\sqrt{K}$, $l\geq j$ is stored in processor j,k, and element (i,j) such that $k=i\bmod\sqrt{K}$, $i=\{0,1,\dots,N-1\}$ is stored in processor k,j.

For the diagonal blocks, the Gauss-Jordan algorithm described previously can be used. For the blocks below the diagonal in the block pivot column, the computations are very similar. Division is performed on the elements of the pivot column, and multiply-add operations on the elements to the right of the pivot column. The operations on blocks to the right of the block pivot column amount to a matrix multi-

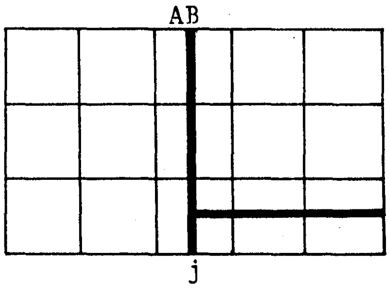


Figure 3—Storage location of elements of a factor, and the corresponding pivot row

plication. This matrix multiplication can be carried out as an outer product algorithm initiating the computation of successive outer products on successive diagonal processors. The same data movement can be used for all block matrix operations. The application of factor $(j-1)\sqrt{K}+r$ starts in processor (r, r) and proceeds in the direction of increasing column index, modulo the number of columns, and in the direction of increasing row index, modulo the number of rows. Clearly, one revolution of the fraction of a pivot row defined by a block matrix suffices for all blocks in a block column. Similarly, one revolution of a fraction of a column suffices for all blocks in a block row. Temporary storage corresponding to the pivot row, or the pivot column is required in each processor row or column, respectively, in order that a fraction need only be communicated once. The processor utilization is very similar to the one depicted in Figure 1.

The computations on different blocks can be pipelined in a manner similar to the case of matrix multiplication. For the triangulation and forward elimination on the right hand side, we derive the following approximate complexity estimate after somewhat tedious calculations.

Proposition 9. The triangulation of an N by N matrix and the forward elimination on R right hand sides can be performed in a time of approximately $(2t_c+t_d)N + \sqrt{K}\lceil N/\sqrt{K}\rceil(\lceil N/\sqrt{K}\rceil + 1)\max(2t_a, t_c)/2 + \sqrt{K}\lceil N/\sqrt{K}\rceil(\lceil R/\sqrt{K}\rceil + (\lceil N/\sqrt{K}\rceil - 1)/2)\max(2t_a, t_c, t_d) + \sqrt{K}\lceil N/\sqrt{K}\rceil(\lceil N/\sqrt{K}\rceil - 1)(3\lceil R/\sqrt{K}\rceil + 2\lceil N/\sqrt{K}\rceil - 1)t_a/3$ on a torus of $\sqrt{K}$ by $\sqrt{K}$ processors.

The estimate in Proposition 9 is rather complex; a few extreme cases may improve the intuitive understanding of it. For $\sqrt{K} = N$, and the time for arithmetic dominating the time for communication, it simplifies to $N(t_d+2t_a)$ for the factorization. Similarly, for the communication dominating the time for arithmetic, it simplifies to $3Nt_c$, also for the factorization. These expressions agree with the corresponding expressions for systolic algorithms. For $\sqrt{K}$ small compared to N and R, the expression is of the form $N(N+R)t_c/\sqrt{K}$, if the communication time dominates. There are N elimination steps, and the number of communications for each step is on the average of order $(N+R)/\sqrt{K}$. If the time for arithmetic operations dominates the time for communication, then the expression takes the form $N(N+R)2t_a/\sqrt{K} + N^2(2N+3R)t_a/(3K)$.

The nature of the expression in Proposition 9 is $\alpha N + N(\beta N + \gamma R)/\sqrt{K} + N^2(\delta N + \epsilon R)/K + \eta$. The speed-up of the highest order term is linear.

Backsubstitution

Accumulation of inner products in place. The choice of algorithm for backsubstitution depends on the number of right hand sides. If there is a large number of right hand sides, then one of the matrix multiplication algorithms described previously can be used effectively. Since the diagonal blocks are diagonalized by the factorization algorithm, the last $\sqrt{K}$ variables for each right hand side are known at the end of this phase. Additional sets of $\sqrt{K}$ variables are computed through matrix multiplications of the form $Y \leftarrow Y - U(i) \times X(i)$, where $U(i)$ is the ith block column of U, $X(i)$ is the ith block row of X, and Y is an i by $\lceil R/\sqrt{K}\rceil$ block matrix. The total number of $\sqrt{K}$ by $\sqrt{K}$ matrix multiplications required to compute the solution X is $\lceil R/\sqrt{K}\rceil\lceil N/\sqrt{K}\rceil(\lceil N/\sqrt{K}\rceil - 1)/2$. The estimated time for the backsubstitution using matrix multiplication algorithm MMT2 is as stated in Proposition 10.

Proposition 10. The solution of an upper triangular system of equations with blocks of size $\sqrt{K}$ by $\sqrt{K}$ and diagonal blocks being diagonalized can be computed on a torus of $\sqrt{K}$ by $\sqrt{K}$ nodes in a time of at most $\lceil R/\sqrt{K}\rceil\lceil N/\sqrt{K}\rceil(\lceil N/\sqrt{K}\rceil - 1)\sqrt{K}\max(2t_a, t_c)/2 + 2(\sqrt{K} - 1)t_c + 2t_a$.

The constant in front of the last communication term depends on which of the previously described multiplication algorithms is used. The speed-up is of order $O(K)$ for R at least of order $O(\sqrt{K})$, but only of order $O(\sqrt{K})$ for few right hand sides.

Accumulation of inner products over space. An algorithm with reduced data movement for few right hand sides is obtained by accumulating inner products over space instead of in-place. At the end of the factorization phase, the last $\sqrt{K}$ x-values are known for each right hand side. The values are computed in the processor column storing the corresponding right hand side. The last $\sqrt{K}$ x-values are used to form a linear combination of the last $\sqrt{K}$ columns of A. The linear combination is subtracted from the corresponding right hand size, and a new set of $\sqrt{K}$ x-values is known. To form the linear combination, each x-value has to intersect with all nonzero matrix elements in the corresponding column.

Assume for simplicity that there is only one right hand side which is stored in processor column 0, and that the last $\sqrt{K}$ columns of U (the upper triangular matrix) are stored in row-major order in processor columns 0 through $\sqrt{K}$. First, rows $k = \{0, 1, \ldots, N-1\}$ of the right hand side are shifted i steps in the direction of increasing column index, $i = k \bmod \sqrt{K}$. This shift requires a time of $(\lceil N/\sqrt{K}\rceil + \sqrt{K}/2)t_c$. For R right hand sides, a shift of length i requires a time of $(\lceil N/\sqrt{K}\rceil \min(i, R) + \max(0, i-R))t_c$. Next, the x-values are communicated along columns, and inner products accumulated along rows. The maximum distance an x-value is communicated is $\lceil\sqrt{K}/2\rceil$. The inner products are accumulated over all $\sqrt{K}$ columns. The communications of x-values and the accumulation can overlap in time to a certain extent. The total time for the accumulation of one set of $\sqrt{K}$ inner products, and updating the right hand side is $t_a+\sqrt{K}(t_a+t_c)$. The number of sets of $\sqrt{K}$ inner products to be computed for the first set of x-values is $\lceil N/\sqrt{K}\rceil - 1$. This number determines the time at which the computations of the next set of $\sqrt{K}$ x-components can be initiated. The number of sets of inner products decreases by one for every set of x-values that are computed. Inner products for each block matrix are computed at the rate of $t_a+\max(t_c,t_a)$, with overlapping communication and computation.

Proposition 11. Solution of the system $AX = B$, where A is upper block triangular of size N by N, has blocks of size $\sqrt{K}$ by $\sqrt{K}$ and diagonalized diagonal blocks, and where B is N by R, by employing a matrix-vector multiplication algorithm with accumulation of inner products over space can be accomplished in a time of approximately $(\lceil N/\sqrt{K}\rceil R + \sqrt{K}/2 - R)t_c + \sum_{j=1}^{\lfloor N/\sqrt{K}\rfloor}\max(t_a + \sqrt{K}(t_a + t_c),(\lceil N/\sqrt{K}\rceil - j)$ $R(t_a+\max(t_a,t_c)))$ on a $\sqrt{K}$ by $\sqrt{K}$ torus.

In order to simplify this expression, we assume that $t_c \geq t_a$. Then, for $\lceil N/\sqrt{K}\rceil R < \sqrt{K}$, the estimated time for backsubstitution is of the form $\alpha N\cdot R/\sqrt{K} + \gamma\sqrt{K} + \beta N$, and for $\lceil N/\sqrt{K}\rceil R \geq \sqrt{K}$, $R \leq \sqrt{K}/2$, the estimated time is of the form $\alpha N^2\cdot R/K + \beta N\cdot R/\sqrt{K} + \gamma\sqrt{K}$.

The speed-up for this backsubstitution algorithm, essentially a matrix vector multiplication algorithm, is linear for K small as well as for $\sqrt{K}$ approaching N. This property is true for all values of R. The complexity of the matrix multiplication algorithm described previously is $\alpha N^2/\sqrt{K} + \beta\sqrt{K}$ for $R \leq \sqrt{K}$. However, at some point as R approaches $\sqrt{K}$, the matrix multiplication algorithm yields a lower complexity. The improved time complexity for small values of R is obtained at the expense of increased storage for temporary variables (approximately $N/\sqrt{K}$ per node). If this is disallowed, the time complexity becomes comparable to that on a mesh. However, the matrix vector multiplication algorithm still has an advantage on a boolean cube.

Gaussian Elimination on a Boolean Cube

For the factorization of a matrix by Gauss-Jordan or Gaussian elimination, the modification of the algorithm for a torus configured ensemble for a boolean cube is identical to the modification described for converting multiplication algorithm MMT3 into MMC4. The GIDD routing has the properties required for correctness (correct order of arrival) when pivoting is performed on the diagonal. Most other operations are indeed very similar to or identical to matrix multiplication from a data structures and control point of view. The complexity estimates for algorithms for torus configured ensembles can be modified to a boolean cube, straightforwardly.

For the matrix-vector multiplication algorithm used in the backsubstitution algorithm for few right hand sides, we note that communication in the form of rotation, one-to-all, and all-to-one (inner product accumulation) is needed. In the section on matrix multiplication, we presented algorithms of complexity order $O(k)$ for all these operations. The complexity estimate for the matrix-vector product algorithm is modified to $(\lceil N/\sqrt{K}\rceil R + k - R)t_c + \sum_{j=1}^{\lceil N/\sqrt{K}\rceil}\max(t_a+2k(t_a+t_c)$, $(\lceil N/\sqrt{K}\rceil - j)R(t_a+\max(t_a,t_c)))$, and the simplifications yield for $t_c \geq t_a$ and $\lceil N/\sqrt{K}\rceil R < 2k$, an estimated time of $\alpha N\cdot R/\sqrt{K} + \gamma k + \beta N\cdot/\sqrt{K}$, and for $\lceil N/\sqrt{K}\rceil R \geq 2k$, $R \leq k$, the $\alpha N^2\cdot R/K + \beta N\cdot R/\sqrt{K} + \gamma k$. With N being the highest order term, the advantage of the matrix-vector multiplication algorithm over the matrix-matrix multiplication algorithm is increased for ensembles configured as boolean cubes.

SUMMARY

We have presented and analyzed the arithmetic and communication complexity of a few algorithms for matrix multiplication and the solution of linear systems of equations on ensembles configured as toruses or boolean cubes. The algorithms have a linear speed-up. Terms that for large problems on small ensembles are of lower order are proportional to the ensemble diameter. The term originates from nonzero communication time between adjacent processors, and is of the minimum order possible.[18] The constant of proportionality varies between two and four.

For boolean cube configured ensembles, algorithms for matrices embedded both by binary and by Gray code encoding of indices are devised and analyzed. One-to-all distribution algorithms are given that terminate after k steps for a k-cube, have distributed control, and deliver a message precisely once to every node.

In our simplified analysis of the algorithms, we have not attempted any detailed optimization of communication strategies, or granularity of operations. Such an optimization must account for communication overhead, and overhead in initiating arithmetic operations. The overhead varies with the degree of parallelism in the architecture, e.g., bit-serial or word parallel, pipeline length, vector instructions, etc.

ACKNOWLEDGMENTS

Thanks go to Ching-Tien Ho for discussions and ideas on the MMT2 algorithm and for his careful proofreading of the manuscript, and to Andrea Pappas for her assistance with the manuscript and preparation of the Figures.

The generous support by the Office of Naval Research under contract N00014-84-K-0043 is also gratefully acknowledged.

REFERENCES

1. Strassen, V. "Gaussian Elimination is Not Optimal." *Numerische Mathematik, 13* (1969), pp. 354–356.
2. Johnsson, S. L. "Highly Concurrent Algorithms for Solving Linear Systems of Equations." *Elliptic Problem Solving, II,* 1983.
3. Chen, M. C. *A Synthesis Method for Systolic Designs.* RR YALEU/DCS/RR-334, Department of Computer Science, Yale University, New Haven, Connecticut, January 1985.
4. Sutherland, I. E., and C. A. Mead. "Microelectronics and Computer Science." *Scientific American,* September 1977, pp. 210–228.
5. Johnsson, S. L. *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures.* YALEU/CSD/RR-367, Department of Computer Science, Yale University, New Haven, Connecticut, February 1985.
6. Flynn, M. J. "Very High-speed Computing Systems." *Proceedings of the IEEE, 12* (1966), pp. 1901–1909.
7. Kung, H. T., and C. L. Leiserson. "Algorithms for VLSI Processor Ar-

rays." In *Introduction to VLSI Systems*. Reading, Mass.: Addision-Wesley, 1980, pp. 271–292.

8. Johnsson, S. L. *Computational Arrays for Band Matrix Equations*. 4287:TR:81, Computer Science Department, California Institute of Technology, Pasadena, May 1981.
9. Cannon, L. E. *A Cellular Computer to Implement the Kalman Filter Algorithm*. Ph.D. thesis, Montana State University, 1969.
10. Johnsson, S. L., U. Weiser, D. Cohen, and A. Davis. "Towards a Formal Treatment of VLSI Arrays." *Proceedings of the Second Caltech Conference on VLSI*. Pasadena, Calif.: California Institute of Technology Computer Science Department, 1981, pp. 375–398.
11. Leiserson, C. E., F. M. Rose, and J. B. Saxe. "Optimizing Synchronous Circuitry by Retiming." Paper presented at Third Caltech Conference on Very Large Scale Integration, Pasadena, California, 1983, pp. 87–116.
12. Dekel, E., D. Nassimi, and S. Sahni. "Parallel Matrix and Graph Algorithms." *SIAM Journal of Computing, 10*-4 (1981), pp. 657–673.
13. Reingold, E. M., J. Nievergelt, and N. Deo. *Combinatorial Algorithms*. Englewood Cliffs, N.J.: Prentice-Hall, 1977.
14. Johnsson, S. L. *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations*. YALEU/CSD/RR-339, Department of Computer Science, Yale University, New Haven, Connecticut, October 1984.
15. Saad, Y. *Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors*. RR YALEU/DCS/RR-348, Department of Computer Science, Yale University, New Haven, Connecticut, January, 1985.
16. Ipsen, I. C. P., Y. Saad, and M. H. Schultz. *Complexity of Dense Linear System Solution on a Multiprocessor Ring*. RR YALEU/DCS/RR-349, Department of Computer Science, Yale University, New Haven, Connecticut, January 1985.
17. Johnsson, S. L. "VLSI Algorithms for Doolittle's, Crout's and Cholesky's Methods." Paper presented at International Conference on Circuits and Computers (ICCC82), September 1982, pp. 372–377.
18. Gentleman, W. M. "Some Complexity Results for Matrix Computations on Parallel Processors." *Journal of the ACM, 25*-1 (1978), pp. 112–115.

Parallel algorithms for bridge- and bi-connectivity on minimum area meshes

by SUSANNE E. HAMBRUSCH
Purdue University
West Lafayette, Indiana

ABSTRACT

I present parallel algorithms for finding the bridge- and bi-connected components of an undirected graph $G = (V, E)$ with n vertices and e edges on an $n^{1/2} \times n^{1/2}$ mesh of processing elements. The algorithms find the bridge-connected components in $O(n^{3/2})$ time for input both in the form of an adjacency matrix and in the form of edges. For bi-connectivity, I show how to achieve $O(n^{3/2})$ time when the input is in adjacency matrix form, and $O(e + n^{3/2})$ time when the input is in the form of edges.

INTRODUCTION

The simple interconnection pattern and the uniform wire length of a mesh of processors appear to make it ideally suited for parallel processing and VLSI computation. Numerous researchers have developed parallel algorithms tailored towards the mesh.[1-7] In this paper, I present parallel algorithms for finding the bridge- and bi-connected components of an undirected graph $G = (V, E)$, $|V| = n$ and $|E| = e$, on a mesh of n processing elements (PEs). It is assumed that every input is read only once, every output is generated once, and every PE contains a constant number of registers of $\log n$ bits each. Under these assumptions, every algorithm finding the bridge-, bi-, and connected components requires at least n PEs. Hence, my algorithms run on a minimum area network. Developing algorithms for minimum area networks is of theoretical interest because of the algorithm technique developed, and of practical interest because area is an expensive resource.

The $n^{1/2} \times n^{1/2}$ mesh receives the input representing G in a number of input waves. It cannot store the inputs of all the input waves for the entire computation. Observe that numerous problems (e.g., directed graph and sorting) cannot be solved on networks with storage capacity less than the length of their input.[3] I consider algorithms in which the graph G is represented by an adjacency matrix as well as algorithms in which G is represented in the form of edges. In the case of an adjacency matrix, the ith input wave consists of the i-row of the matrix; in the case of edges, the ith input wave consists of n arbitrary edges of G. In the ith input wave PE_j, $1 \leq j \leq n$, receives exactly one input (which is either the bit a_{ij} or an edge (x_j, y_j)). I assume that the input waves are received in a when-indeterminate mode[8] (i.e., the time at which the ith input wave is read may depend on the data). I show how to determine the bridge-connected components in $O(n^{3/2})$ time for both adjacency matrix and edge input. For bi-connectivity, I show how to achieve $O(n^{3/2})$ time when the input is in adjacency matrix form, and $O(e + n^{3/2})$ time when the input is in the form of edges.

Algorithms for graph problems on parallel models with enough PEs and memory to store a representation of the graph explicitly during the entire computation have been studied extensively for a variety of parallel models.[1, 9-15] The issues involved when only part of the input is available at any time during the algorithm, and where this input has to be processed (i.e., irrelevant inputs are discarded) before the next input wave can be read, are quite different. Lipton and Valdes[16] and Hochschild et al.[17] consider binary tree networks with n leaves for solving graph problems with adjacency matrix input. The algorithms in Hochschild et al. require $\log n$ registers per PE, and the bi-connectivity algorithm in Lipton and Valdes has to read the adjacency matrix twice. I have previously used the model in this paper and described algorithms on $O(n)$ area meshes for finding the connected components for both forms of input.[18]

In my algorithms, I assume that the n PEs, $PE_1, \ldots, PE_n$, are arranged in snake-like row-major order (i.e., PE_i is directly connected to PE_{i-1} and PE_{i+1}, provided they exist). This assumption is for convenience only, and the time bounds hold when other standard indexing schemas are used. The time bounds of the algorithms are further independent of how the ith input wave is being input. If only $n^{1/2}$ PEs on the boundary of the mesh can perform the I/O, the additional $n^{1/2}$ time needed to read the ith input wave does not change the time bounds.

The algorithms for bridge-connectivity associate with every vertex of the graph a connected and a bridge-connected component number. The output consists of a bridge-connected component number for every vertex. This number does not correspond to the index of the smallest vertex in this bridge-connected component (how it is determined is described in the section *Bridge Connectivity*), but our algorithms can easily be modified to produce output of this form. The algorithms for bi-connectivity also number the bi-connected components. The output lists for every vertex the bi-connected components containing this vertex. Observe that bi-connectivity induces an equivalence relation on the edges, while bridge-connectivity induces one on the vertices.

My algorithms process the ith input wave completely before reading the $(i + 1)$th input wave. Processing an input wave consists of determining which inputs are irrelevant (and discarding them) and incorporating the relevant input (i.e., the input which contains new information about the bridge- (resp. bi-) connected components) into the data structures used on the mesh. The elements of the data structures are distributed among PEs in a way that allows for fast data movement and thus fast processing of an input wave.

BRIDGE-CONNECTIVITY

In this section, I present an algorithm for finding the bridge-connected components on a two-dimensional mesh of $O(n)$ area in time $O(n^{3/2})$. I first give the algorithm for input in the form of an adjacency matrix, and then describe the modifications to be done when the graph is represented in the form of edges.

Let us start with an informal description of the approach used in the bridge-connectivity algorithm. Every vertex i has two integers, C_i, the current component number of i, and B_i, the current bridge-connected component number of i, associated with it, $1 \le i \le n$. These two entries are stored in PE_i in the mesh. Initially, $B_i = C_i = i$, $1 \le i \le n$. The algorithm puts two vertices in the same bridge-connected component iff there exist two edge-disjoint paths between them. In order to determine this, the algorithm stores in the mesh the edges (at most $n - 1$) that have so far caused the merge of two connected components. These edges form a forest where every tree represents a connected component and is called a connectivity tree.

When the ith row of the adjacency matrix is read, PE_j reads the entry a_{ij}, $1 \le j \le n$. If $a_{ij} = 1$ and $C_i \neq C_j$, the connectivity tree containing vertex i and the one containing vertex j are connected through the edge (i, j) (i.e., the connected components C_i and C_j are merged. The edge (i, j) is recorded in the mesh as an edge of the newly formed connectivity tree. If $a_{ij} = 1$ and $C_i = C_j$, the edge (i, j) forms a cycle in the connectivity tree representing the connected component C_i, and the algorithm determines the bridge-connected components merged by the edge (i, j). If $B_i \neq B_j$, all the bridge-connected components that contain at least one vertex on the path from i (resp. j) to the lowest common ancestor of i and j in the connectivity tree (containing vertices i and j) form a new bridge-connected component. The information about the connectivity tree has to be organized such that these vertices can be determined easily.

I next describe the organization of the entries of the connectivity trees. The entries representing the connected component CX have the form of edges of a rooted tree. The root of the tree is vertex CX. More precisely, every connectivity tree entry is a six-tupel (CX, X, PX, DX, BX, DBX) where

CX is the component number of the vertex X,
PX is the parent node of X in the connectivity tree with root CX,
DX is the depth of X in the connectivity tree CX,
BX is the bridge-connected component number of X; BX is always equal to the vertex in BX that has the smallest depth (i.e., is the closest to the root of the connectivity tree),
DBX is the depth of the vertex BX.

In Figure 1, the broken undirected edges indicate edges that merged bridge-connected components. Connectivity tree entries are stored in the mesh, sorted according to the component numbers CX. Entries belonging to the same connectivity tree are kept sorted according to their depth DX in the tree.

Initially, PE_i contains the connectivity tree entry $(i, i, 0, 0, i, 0)$, but in the later stages of the algorithm, there is no relation between the connectivity entry stored in PE_i and vertex i. In addition to the connected component register C_i and bridge-connected component register B_i, two other registers in PE_i are associated with vertex i, D_i, which contains the depth of vertex i in the connectivity tree with root C_i, and NR_i, which contains the number of vertices in the connectivity tree C_i. Information about vertex i is thus kept both in PE_i, and in the connectivity tree entry for vertex i. Auxiliary registers are introduced when needed.

In the description of the implementation of the algorithm, I assume that the following subroutines are available:

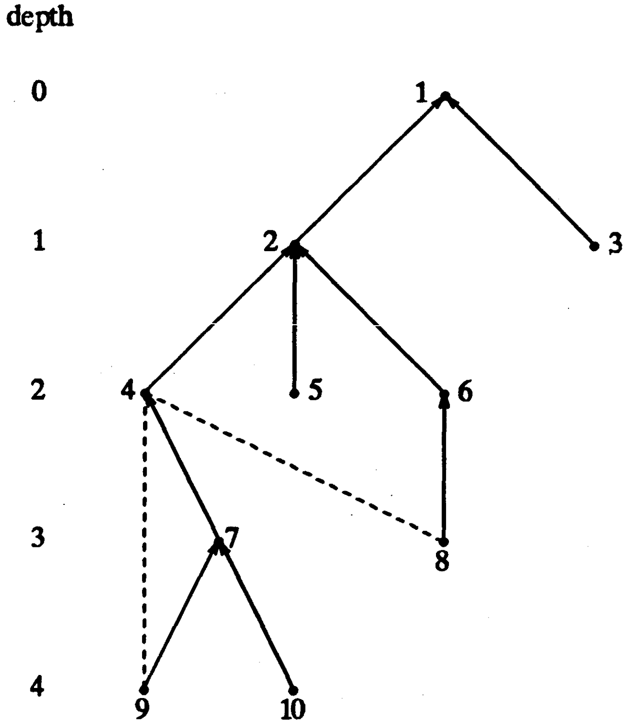


Figure 1—A connectivity tree with vertices 2, 4, 6, 7, 8, and 9 in the bridge-connected component 2; the connectivity entry for vertex 9 is (1,9,7,4,2,1)

Random-Access-Read (RAR): PE_i requests the content of register R_j of PE_j and stores it in register R_i. This operation is denoted by $R_i := R_j$, or $R := R_j$ if the value of i is clear from the context. Note that different PEs can request data from the same PE.
SORT: Specified data items in the mesh are sorted in increasing order.
PACK: k PEs in the mesh are "flagged." PACK moves specified data in the flagged PEs (while maintaining their original order) into lower numbered PEs (i.e., the data in the ith flagged PE is moved into PE_i).

All of the above subroutines can be implemented to run in $O(n^{1/2})$ on a mesh of n PEs.[3,7,19]

Combining the Connectivity Trees

After the PEs have read the ith row of the matrix, the values of C_i, NR_i, and D_i stored in PE_i are distributed to every PE in the mesh. If there is an edge from vertex i to j, PE_j sets registers as shown in Figure 2.

The entries $(I, J, CI, CJ, NRI, NRJ, DI, DJ)$ that are in PEs with $a_{ij} = 1$ and $CI \neq CJ$ are called the tree-combining entries. The algorithm sorts the tree-combining entries in increasing order according to CJ. After the sort, the algorithm sets a flag in PE_1, and in every PE_j that contains a tree-combining entry for which the value of CJ differs from the value of CJ in PE_{j-1}. It then calls routine PACK. Assume $PE_1, \ldots, PE_p$ contain the flagged tree-combining entries

```
for all $PE_j$, $1 \le j \le n$ pardo
    $CI := C_i$
    $NRI := NR_i$
    $DI := D_i$
    if $a_{ij} = 1$ then $J := j$
                    $CJ := C_j$
                    $NRJ := NR_j$
                    $DJ := D_j$
odpar
```

Figure 2—Setting registers at the beginning of the ith iteration

$(I, J, CI, CJ, NRI, NRJ, DI, DJ)$ after PACK. These entries represent p edges that connect $p+1$ connectivity trees, namely $CI, CJ_1, \dots, CJ_p$. Note that, throughout the description of the algorithms, I refer to the value stored in a register R_i simply as R_i. The next step of the algorithm is to combine the $p+1$ connectivity trees into one. Since the connectivity tree entries are stored as edges of a rooted tree, we have to "reroot" some connectivity trees. When a non-root vertex of a connectivity tree becomes the new root, the edges on the path from the old root to the new have to be reversed, and the depth of all vertices in the connectivity tree has to be updated.

The rerooting of the connectivity trees is a potentially time consuming procedure. In order to achieve the claimed time bound, the algorithm never reroots the connectivity tree containing the largest number of vertices (among all the other trees to be rerooted). Thus, before the start of the rerooting process, the algorithm rearranges the tree-combining entries so that the entry stored in PE_1 has the largest NRJ value (i.e., $NRJ_1 = \max\ \{NRJ_1, \dots, NRJ_p\}$. Recall that $CI, CJ_1, \dots, CJ_p$ are the connectivity trees to be combined, and that the entries CI, NRI, and DI have the same value in the p PEs.

1. If $NCI_1 \ge NCJ_1$, then the connectivity tree CI containing vertex I is not rerooted. In the connectivity trees $CJ_1, \dots, CJ_p$ vertices $J_1, \dots, J_p$ are made the new roots at a depth of $DI + 1$ (see Figure 3(a)).
2. If $NCI_1 < NCJ_1$, then the tree CJ_1 containing vertex J_1 is not rerooted. In the connectivity tree CI, vertex I is made the new root at depth $DJ_1 + 1$, and in the trees $CJ_2, \dots, CJ_p$, the vertices $J_2, \dots, J_p$ are made the new roots at a depth of $DJ_1 + 2$ (see Figure 3(b)).

I next discuss the rerooting process for the first case (i.e., $NCI \ge NCJ_1$). The second case is handled in a similar fashion. Every tree-combining entry creates a reroot entry $(I, J, CI, CJ, ND)_r$, where ND, the new depth of vertex J, is equal to $DI + 1$, and vertex J will be the new root of the vertices in the connectivity tree CJ. (Note that the subscript r is used to indicate a reroot entry, not a PE.) Every one of the p reroot entries is sent to the PE that contains the connectivity entry for vertex J (i.e., to the PE containing the entry (CX, X, PX, DX, BX, DBX) with $CX = CJ$ and $X = J$). Observe that the PE creating the reroot entry does not "know" the position of this connectivity entry. The position is determined by sorting all the connectivity tree entries belonging to vertices that are roots, and the p rerooting entries according to the component numbers. By doing so, every reroot entry determines the position of the root of its connectivity tree in $O(n^{1/2})$ time. Once every reroot entry has been sent to the PE containing the root, it locates the connectivity entry corresponding to vertex J in $O(n^{1/2})$ time (recall that the connectivity entries of every tree CX are sorted according to their depth). Now, the actual rerooting of connectivity trees CX starts, and the p connectivity trees are rerooted in parallel.

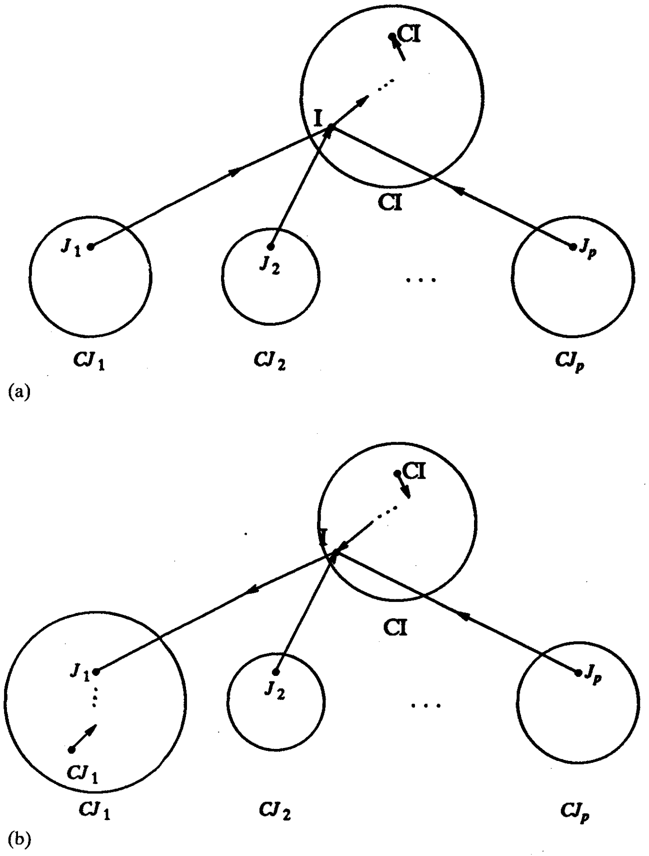


Figure 3—(a) CI contains the largest number of vertices; (b) CJ_1 contains the largest number of vertices

The rerooting of every tree CX works in two phases. The first phase reverses the edges on the path from vertex X to the root CX (and also updates connectivity tree entries), and the second phase updates the depth of the vertices in the subtrees rooted on a vertex on the path from X to CX. Both phases use $O(n^{1/2} + m)$ time, where m is the number of vertices in tree CX.

I now describe the implementation of the first phase in more detail. Let (CX, X, PX, DX, BX, DBX) be a connectivity tree entry in PE_k that received the reroot entry $(I, J, CI, CJ, ND)_r$.

1. If $X \neq J$, PE_k sends the reroot entry to PE_{k-1} without changing it or its own registers.
2. If $X = J$, PE_k updates its connectivity tree entry by setting $CX := CI$, $PX := I$, and $DX := ND$. Then, PE_k creates the update entry $(J, CI, CJ, ND)_u$ with $ND = ND + 1$ and the value of registers J, CI, and CJ as in the reroot entry. The update entry remains stored in PE_k until it is activated in the second phase. Next, PE_k

changes the reroot entry as follows. If vertex J (which in this case is equal to vertex X) is not the root (i.e., $X \neq CX$), PE_k sends the reroot entry $(I, J, CI, CJ, ND)_r$ with $I = X, J = PX, ND = ND + 1$, CI and CJ unchanged, to PE_{k-1}. If vertex X is the root, the second phase starts.

After the first phase, every PE containing a connectivity entry of a vertex that is incident to an edge of the tree which got reversed contains an update entry $(J, CI, CJ, ND)_u$. The goal of the second phase is to send every update entry $(J, CI, CJ, ND)_u$ to the children of vertex J (excluding the child that is now a parent), and to change the depth in the connectivity entry of the children to ND. Every child will then create its own update entry to be sent to its children, etc. Every PE containing a connectivity tree entry thus creates (or already contains) exactly one update entry.

I now describe how to implement the second phase in $O(n^{1/2} + m)$ time. If every update entry originally in PE_k is sent (independent of the other update entries) to PE_{k+1}, $PE_{k+2}, \ldots$, and if the PEs (which contain the connectivity entries of children) create their own update entries (which are also sent to higher numbered PEs), the algorithm encounters *congestion* problems. Thus, the algorithm does the following. The update entry in the root is activated first (i.e., if the connectivity entry of the root is in PE_k, PE_k sends its update entry to $PE_{k+1}, PE_{k+2}, \cdots$). Assume PE_l receives an update entry $(J, CI, CJ, ND)_u$.

1. If $PX_l \neq J$ (i.e., the connectivity entry in PE_l does not belong to a child of vertex J), PE_l sends the update entry to PE_{l+1}.
2. If $PX_l = J$, the algorithm sets register DX_l (of the connectivity entry) equal to ND, CX_l equal to CI, and it creates a new update entry (J^*, CI^*, CJ^*, ND_*) with $J^* = X$, $CI^* = CI$, $CJ^* = CJ$, and $ND^* = ND + 1$. PE_l sends the old update entry to PE_{l+1}, and keeps the newly created one until it is activated. The newly created update entry in PE_l is activated after the update entry created in PE_{l-1} passes through PE_l.

It is easy to see that this technique does not run into congestion problems and that after $O(m)$ time, where m is the number of vertices in the tree, every connectivity tree entry contains the new values.

From the above discussion it follows that the p connectivity trees can be rerooted in $O(n^{1/2} + m)$ time, where m is the number of vertices in the second largest connectivity tree involved. Before proceeding with the next major step of the algorithm, the determining and merging of bridge-connected components, we have to update the entries about vertex k in PE_k, $1 \leq k \leq n$. The number of vertices in the new connectivity tree with root CI (resp. CJ_1) can be computed in $O(n^{1/2})$ time using the p tree-combining entries. Every vertex k in $CI, CJ_1, \ldots, CJ_p$ can update its component number C_k and the value NR_k to the new values in $O(n^{1/2})$ time (by using SORT twice). A write operation initiated by the connectivity entries updates the depth registers D_k in every PE_k in $O(n^{1/2})$ time.

Merging Bridge-Connected Components

After the connectivity trees have been combined, every PE_j with $a_{ij} = 1$ has $C_i = C_j$, where C_i is the updated connected component number. If the edge (i, j) were used as a tree-combining edge, we set a_{ij} to 0. Next, every PE_j obtains the values B_i and D_i, and if $B_i = B_j$ also sets a_{ij} to 0, $1 \leq j \leq n$. Every remaining PE_j with $a_{ij} = 1$ and $B_i \neq B_j$ contains an edge that merges bridge-connected components and creates the bridge entry $(I, J, CI, BI, BJ, DI, DJ)_b$. The values of a bridge entry are set similar to the code shown in Figure 2.

While the algorithm determines the bridge-connected components merged by a bridge entry, only the section of the mesh containing the connectivity tree entries of tree CI is used. The algorithm can thus process bridge entries of different connectivity trees simultaneously. Since doing so does not affect the worst case time performance, I will not discuss this possibility in more detail. When the algorithm chooses one bridge entry $(I, J, CI, BI, BJ, DI, DJ)_b$, it follows the path from vertex I to the lowest common ancestor of I and J, referred to as lca(I, J), and the path from vertex J to lca(I, J). It marks all bridge-connected components encountered on these two paths as to be merged into one. I now describe in more detail how a bridge entry is processed in $O(bn^{1/2})$ time, where b is the number of bridge-connected components merged by the edge (I, J).

The bridge entry $(I, J, CI, BI, BJ, DI, DJ)_b$ is sent to the PE containing the connectivity entry of the root of connectivity tree CI. Let PE_f be this PE. At PE_f, the bridge entry is split up into two entries, $(I, CI, BI, DI)_b$ and $(J, CI, BJ, DJ)_b$, which will from now on be called the bridge entries. If $DI = DJ$, then both bridge entries are sent from PE_f to the PE containing the connectivity entry of vertex I and J, respectively. If $DI < DJ$, then only the bridge entry containing vertex J is sent, and if $DI > DJ$, then only the bridge entry containing vertex I is sent. This ensures that we move in the connectivity tree from I and J towards the lca(I, J) "at the same pace."

I next describe what the bridge entry $(I, CI, BI, DI)_b$ does. The action for the bridge entry for J is analogous to it. Assume that the connectivity tree entry for vertex I is in PE_{k1} (i.e., PE_{k1} contains entry $(CX_{k1}, X_{k1}, PX_{k1}, DX_{k1}, BX_{k1}, DBX_{k1})$ with $X_{k1} = I$ and $CX_{k1} = CI$, $DX_{k1} = DI$, and $BX_{k1} = BI$). PE_{k1} sets a flag to indicate that it contains a bridge-connected component to be used in the merge.

1. If $BX_{k1} = X_{k1}$, then PE_{k1} sends its bridge entry to the PE containing the connectivity entry of vertex PX_{k1}, the parent of vertex X_{k1}.
2. If $BX_{k1} \neq X_{k1}$, then PE_{k1} sends its bridge entry to the PE containing the connectivity entry of vertex BX_{k1} which is at depth DBX_{k1}. Note that by sending the bridge entry to the PE containing the entry of BX_{k1}, the algorithm never traverses edges that are in already existing bridge-connected components.

Let PE_{k2} be the PE receiving the bridge entry from PE_{k1}. The bridge entry can be sent from PE_{k1} to PE_{k2} in $O(n^{1/2})$ time. At

PE_{k2}, the bridge entry $(I, CI, BI, DI)_b$ is updated to $I = X_{k2}$, $BI = BX_{k2}$, and $DI = DX_{k2}$. The udpated bridge entry is sent to PE_f. When PE_f receives the updated bridge entry (resp. entries), it checks whether the bridge-connected component containing the lca(I, J) has been reached.

1. If $BI \neq BJ$, then PE_f sends out either one or both bridge entries (depending on the current depth in the bridge entries).
2. If $BI = BJ$, the lowest common bridge-connected component has been reached, and PE_f sets $BNEW = BI$. $BNEW$ will be the new bridge-connected component number of all the vertices in bridge-connected components that received a flag, and the updating of bridge-connected component entries begins.

I now describe the final updating of the entries. The algorithm calls routine PACK, which places the flagged connectivity entries containing bridge-connected components to be merged in $PE_1, \dots, PE_s$. Let $B_{i_1}, \dots, B_{i_z}$ be these bridge-connected components. $BMIN_f$ is made the new bridge-connected component number of all the vertices in $B_{i1}, \dots, B_{i_z}$. This change has to be recorded in the bridge-connected component number B_k of vertex k in PE_k, and in the bridge-connected component numbers in the connectivity entries containing vertex k. Furthermore, the entry DBX in the connectivity entries belonging to vertices of flagged bridge-connected components has to be updated. Note that the new value of DBX of all the vertices involved in the merging is the depth of vertex $BNEW_f$. The updating of all these entries can be done in $O(n^{1/2})$ time.

Theorem 1: The bridge-connected components can be found in time $O(n^{3/2})$ on a two-dimensional mesh of $O(n)$ area when the graph is given in the form of an adjacency matrix.

Proof: The correctness of the algorithm follows from the preceding discussion. The time bound is obtained as follows. The time not spent on the combining of connectivity trees or the merging of bridge-connected components is $O(n^{1/2})$ for each row of the adjacency matrix. We have shown that the time used to combine and reroot connectivity trees is $O(n^{1/2} + m)$ in each iteration where m is the number of vertices in the second largest component to be merged in the ith iteration. In the worst case we combine and reroot connectivity trees of the same size, and we combine only two connectivity trees in each iteration (i.e., we combine two trees of $n/2$ vertices each in the nth iteration, two trees of $n/4$ vertices each in the $(n - 1)$th and $(n - 2)$th iteration, etc.) Thus, the total time spent on combining connectivity trees is

$$O((n/2 + n^{1/2}) + 2(n/4 + n^{1/2}) + \\ 4(n/8 + n^{1/2}) + \cdots + n/2(1 + n^{1/2})),$$

which is $O(n^{3/2})$. The overall time spent on the processing of bridge entries and the merging of bridge-connected components is also $O(n^{3/2})$, since at most $n - 1$ bridge-connected components can be merged. Hence, the total time of our algorithm is $O(n^{3/2})$. □

Our algorithm can be extended to find the bridge-connected components in time $O(n^{3/2})$ when the input is given in the form of edges. The overall structure of the algorithm and the entries created during the computation remain the same. Observe that now PE_k, $1 \le k \le n$, reads an arbitrary edge (I, J) and that the connected component number of vertex I (resp. J) is in PE_I (resp. PE_J). While the merging of bridge-connected components is done by processing the bridge entries one by one as before, the situation for combining connectivity trees is different. When the graph is given in the form of an adjacency matrix, the edges that merge connectivity trees at the ith iteration represent a connected graph with no transitive edges (see Figure 3); when the graph is given in the form of edges, this is no longer true. The edges between connectivity trees can now represent a graph that is not necessarily connected and that can contain transitive edges. But in order to achieve $O(n^{3/2})$ time, the connectivity trees do not have to be combined in parallel. We only have to make sure that the connectivity tree with the largest number of vertices is never rerooted. Hence, by making this step more sequential, the following result is obtained.

Theorem 2: The bridge-connected components can be found in time $O(n^{3/2})$ on a two-dimensional mesh of $O(n)$ area when the graph is given in the form of edges.

BI-CONNECTIVITY

In this section, I describe an algorithm that determines the bi-connected components of an undirected graph on an $O(n)$ area mesh in time $O(n^{3/2})$ when the input is given in the form of an adjacency matrix. As in the algorithm for bridge-connectivity, associate with every vertex a connected component number, and record the edges that caused the merge of two connected components as entries of connectivity trees. Bi-connected component numbers are used to record the bi-connectivity information obtained about the graph so far. Since one vertex can be in more than one (and at most $n/2$) bi-connected components, PE_i cannot be used to store the bi-connectivity numbers of vertex i. The connectivity trees help to determine the bi-connected components, and the algorithm puts two vertices in the same bi-connected component iff it finds two vertex-disjoint paths between them.

The algorithm records in PE_i the entries C_i, D_i, and NR_i associated with vertex i as defined in the previous algorithm. It records the bi-connectivity information in the form of bi-number entries. Every such entry is a four-tupel consisting of

1. A vertex,
2. A bi-connected component number (that the vertex is currently in),
3. The vertex in the same bi-connected component number that has smallest depth in the connectivity tree, and
4. The depth of this vertex.

The vertex at the smallest depth in the connectivity tree cannot be used as the bi-connected component number, because this vertex could be in more than one bi-connected component. Bi-connected component numbers are now assigned as follows. PE_1 contains a register *NUMB*, which is initially set to 1. Every time a new bi-connected component is formed, it gets the number equal to the current value of *NUMB*, and *NUMB* is increased by 1. Because every time *NUMB* is increased, at least two bi-connected components get merged, the final value of *NUMB* is at most $n - 1$.

Every PE contains registers to store up to two bi-number entries, namely registers $(I1, BI1, OI1, DOI1)$ and $(I2, BI2, OI2, DOI2)$. We refer to these two sets of registers as (I^*, BI^*, OI^*, DOI^*). It is easy to show that in any graph there can be at most $(3n - 3)/2$ bi-number entries. Thus, two per PE are sufficient. At some time during the algorithm, the bi-number entries are sorted according to the vertices; at other times, the entries they are sorted according to the bi-connected component numbers. The bi-number entries are stored in packed form (i.e., the entries in PE_i are filled after the $2(i-1)$ bi-number entries in $PE_1, \ldots, PE_{i-1}$ have been filled). Initially, the mesh contains the n bi-number entries $(i, 0, i, 0)$, $1 \leq i \leq n$.

The combining and rerooting of the connectivity trees, and the merging of connected components is done as in the bridge-connectivity algorithm. Note that a connectivity tree entry is now a four-tupel (CX, X, PX, DX), and that after the rerooting process, the DOI^* component in the bi-number entries needs to be updated.

After the combining and rerooting of the connectivity trees, every PE_i with $a_{ij} = 1$ and edge (i, j) not used in the connected component merging process contains an edge entry (I, J, CI, DI, DJ) with $I = i$ and $J = j$. Next, the algorithm finds one edge entry that forms new bi-connected components. It determines in $O(n^{1/2})$ time either an edge entry that causes the merge of at least two bi-connected components, or it concludes in $O(n^{1/2})$ time that none of the up to n edge entries merges bi-connected components. An edge (I, J) merges bi-connected components if no bi-connected component contains both I and J. In terms of bi-number entries and edge entries, this condition is stated as follows. The edge entry (I, J, CI, DI, DJ) merges bi-connected components if for all bi-number entries $(I_k^*, BI_k^*, OI_k^*, DOI_k^*)$ and $(I_l^*, BI_l^*, OI_e^*, DOI_l^*)$ with $I_k^* = I$ and $I_l^* = J$, $BI_k^* \neq BI_l^*$ holds. It is easy to check this condition in $O(n^{1/2})$ time for a given edge entry. How one edge entry satisfying the condition is found, or how it is determined that no edge entry satisfies it in $O(n^{1/2})$ time is described next.

Selecting an Edge Entry

The algorithm adds a mark register $MARK_k^*$, $1 \leq k \leq n$, to every bi-number entry. $MARK_k^*$ is initially set to 0. The selection of an edge entry is done in three stages. In the first stage, the algorithm sets the mark registers in all bi-number entries of vertices adjacent to vertex I to 1 (i.e., it sets $MARK_k^* = 1$ in every bi-number entry $(I_k^*, BI_k^*, OI_k^*, DOI_k^*, MARK_k^*)$ with $I_k^* = J_l$, where $(I_l, J_l, CI_l, DI_l, DJ_l)$ is an edge entry. This step is implemented in $O(n^{1/2})$ time by sorting the bi-number entries according to the vertices, then sending every edge entry $(I_l, J_l, CI_l, DI_l, DJ_l)$ to the lowest indexed PE_k containing a bi-number entry with $I_k^* = J_l$, and propagating this edge entry to higher-numbered PEs.

In the second stage, the algorithm sets the mark registers in bi-number entries $(I_k^*, BI_k^*, OI_k^*, DOI_k^*, MARK_k^*)$ with $MARK_k^* = 1$ to 2 if there exists a bi-number entry $(I_l^*, BI_l^*, OI_l^*, DOI_l^*, MARK_l^*)$ with $I_l^* = i$ and $BI_k^* = BI_l^*$. This step is implemented in $O(n^{1/2})$ time by sorting the bi-number entries according to the bi-connected component numbers, and letting every bi-number entry with $I_l^* = i$ mark the entries with $BI_k^* = BI_l^*$.

In the third and final stage in the selection of an edge entry, the algorithm sorts the bi-number entries according to the vertices. It then selects, in $O(n^{1/2})$ time, among all edge entries (I, J, CI, DI, DJ) for which no bi-number entry corresponding to vertex J has the mark register set to 2, an arbitrary one. If no such edge entry is found, the ith iteration of the algorithm is completed and row $i + 1$ of the adjacency matrix is read next.

Merging of Bi-Connected Components

After an edge entry, say (I, J, CI, DI, DJ), has been selected, the algorithm merges bi-connected components. The basic concept of the merging is similar to the one used in the algorithm for bridge-connectivity. The algorithm follows the paths from vertices I and J to the lowest common ancestor of I and J in the connectivity tree CI. Obviously, all the vertices on the two paths belong to one bi-connected component. In addition, include a bi-connected component that contains at least two vertices that are on these two paths.

The data movement for determining the bi-connected components to be merged is similar to the one for bridge-connectivity; I only point out some of the differences. The bi-connectivity information about a vertex is not stored in the connectivity tree entry; it has to be "looked up" in bi-number entries. This adds an additional $O(n^{1/2})$ time for traversing every edge on the paths. Existing bi-connected components encountered on the paths are only included if they contain at least two vertices that are on the path to the lca(I, J). The algorithm uses the depth entry DOI^* of the vertex OI^* of the bi-connected component BI^* to avoid traversing more than one edge in the bi-connected component BI^*. I leave the implementation details to the reader. It follows that the time for processing one edge entry is $O(mn^{1/2})$, where m is the number of bi-connected components that get merged by the edge (I, J). Again, at most $n - 1$ bi-connected components can get merged, and the overall time of the algorithm is $O(n^{3/2})$.

Theorem 3: The bi-connected components can be found in time $O(n^{3/2})$ on a two-dimensional mesh of $O(n)$ area when the graph is given in the form of an adjacency matrix.

Proof: Similar to the proof of Theorem 1. □

I next describe how to modify the above algorithm to find the bi-connected components in $O(e + n^{3/2})$ time when the graph is given in the form of edges. Recall that for bridge-connectivity, $O(n^{3/2})$ time can be achieved for input in the form of edges. The time-critical step in the bi-connectivity algorithm is selecting an edge entry that merges bi-connected components (or deciding that none exists) efficiently. When the idea of marking bi-number entries is applied to an arbitrary set of edges instead of edges adjacent to vertex i, the irregularity of the input causes an increase in the time complexity.

I now describe the difficulties that arise and give an informal outline how to process $O(n^{1/2})$ edge entries in $O(n)$ time. Let (x_i, y_i) be n edges that do not merge connected components and assume they are stored in $PE_1, \ldots, PE_n$ of the mesh. Let the number of bi-number entries containing vertex x_i be less than or equal to the number of bi-number entries containing vertex y_i. If vertex x_i is in the bi-connected components $B_i^1, \ldots, B_i^{l_i}$, form the triples (x_i, y_i, B_i^k), $1 \leq k \leq l_i$. Then check for every triple (x_i, y_i, B_i^k) whether or not vertex y_i is in the bi-connected component B_i^k. Unfortunately, we cannot create all the triples of the n edges at once, because n edges can result in $O(n^{3/2})$ triples in the worst case.

Consider a graph in which every one of the vertices $x_1, \ldots, x_{n^{1/2}}$ is currently in $n^{1/2}$ bi-connected components and every one of the vertices $y_1, \ldots, y_{n^{1/2}}$ is currently in $n^{1/2}$ bi-connected components. Let the next input sequence contain the edges (x_i, y_j), $1 \leq j \leq n^{1/2}$, $1 \leq i \leq n^{1/2}$. Then every edge (x_i, y_j) creates $n^{1/2}$ triples, and altogether $n^{3/2}$ triples are created. This bound is achieved by a graph that has $n = 4k^2$ vertices in which vertices $x_1, \ldots, x_k$, and $y_1, \ldots, y_k$ are on one cycle, and in which every vertex x_i (resp. y_i) is in k bi-connected components (namely the cycle and k "triangles." The edges (x_i, y_i) form $k^3 = n^{3/2}/4\sqrt{2}$ triples, but no new bi-connected components.

In the selection of an edge entry, we handle a batch of $n^{1/2}$ edges at a time. For $n^{1/2}$ edge entries, we form the triples as outlined above (note that at most $O(n)$ triples can be created), and then select an edge entry by marking bi-number entries similar to the marking step for input in the form of an adjacency matrix. Once an edge entry has been selected and bi-connected components merged, the next edge entry is selected from the current batch of $n^{1/2}$ edges in $O(n^{1/2})$ time. Thus, the total time for processing n edge entries (not counting the time to merge bi-connected components) is $O(n)$. The overall time spent in selecting edge entries is $O\left(\frac{e}{n} n\right) = O(e)$. The time spent in the other steps of the algorithm remains the same. We can thus state the following.

Theorem 4: The bi-connected components can be found in time $O(e + n^{3/2})$ on a two-dimensional mesh of $O(n)$ area when the graph is given in the form of edges.

ACKNOWLEDGMENT

This work was supported in part by the Office of Naval Research Contract N00014-84-K-0502.

REFERENCES

1. Atallah, M. J., and S. R. Kosaraju. "Graph Problems on a Mesh-Connected Processor Array." *Journal of the ACM,* 31 (1984), pp. 649–667.
2. Guibas, L. J., H. T. Kung, and C. D. Thompson. "Direct VLSI Implementation for Combinatorial Algorithms." *Proceedings of the Caltech Conference on VLSI Technical Design and Fabrication.* Pasadena, Calif.: California Institute of Technology, 1979.
3. Hambrusch, S. E. "The Complexity of Graph Problems on VLSI." Ph.D. thesis, Pennsylvania State University, August 1982.
4. Kung, H. T., and C. E. Leiserson. "Systolic Arrays for VLSI." In C. Mead and L. Conway (eds.), *Introduction to VLSI Systems.* Reading, Mass.: Addison-Wesley, 1980.
5. Miller, R., and Q. F. Stout. "Computational Geometry on a Mesh-Connected Computer." *Proceedings of International Conference on Parallel Processing.* Bellaire, Mich., IEEE Computer Society Press, 1984, pp. 66–73.
6. Miller, R., and Q. F. Stout. "Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer." *IEEE Transactions on Pattern Analysis and Machine Intelligence,*
7. Thompson, C., and H. Kung. "Sorting on a Mesh-Connected Parallel Computer." *Communications of the ACM,* 20-4 (1977), pp. 263–271.
8. Ullman, J. D. *Computational Aspects of VLSI.* Rockville, Md.: Computer Science Press, 1984.
9. Dekel, E., D. Nassimi, and S. Sahni. "Parallel Matrix and Graph Algorithms." *SIAM Journal on Computing,* 10 (1981), pp. 657–675.
10. Hirschberg, D. S., A. K. Chandra, and D. V. Sarwate. "Computing Connected Components on Parallel Computers." *Communications of the ACM,* (1979), pp. 461–464.
11. Ja'Ja', J., and J. Simon. "Parallel Algorithms in Graph Theory: Planarity Testing." *SIAM Journal on Computing,* 11 (1982), pp. 314–328.
12. Nassimi, D., and S. Sahni. "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer." *SIAM Journal on Computing,* 9 (1980), pp. 744–757.
13. Savage, C., and J. Ja'Ja'. "Fast, Efficient Parallel Algorithms for Some Graph Problems." *SIAM Journal on Computing,* 10 (1981), pp. 682–691.
14. Shiloach, Y., and U. Vishkin. "An $O(\log n)$ Parallel Connectivity Algorithm." *Journal of Algorithms,* 3 (1982), pp. 57–67.
15. Tsin, Y. H., and F. Y. Chin. "Efficient Parallel Graph Algorithms for a Class of Graph Theoretic Problems." *SIAM Journal on Computing,* 13 (1984), pp. 580–599.
16. Lipton, R. J., and J. Valdes. "Census Function: An Approach to VLSI Upper Bounds." *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science.* Nashville, Tenn., IEEE Computer Society Press, 1981, pp. 13–22.
17. Hochschild, P. H., E. W. Mayr, and A. R. Siegel. "Techniques for Solving Graph Problems in Parallel Environments." *Proceedings of 24th IEEE FOCS Conference.* Tucson, Ariz., IEEE Computer Society Press, 1983, pp. 351–359.
18. Hambrusch, S. E. "VLSI Algorithms for the Connected Component Problem." *SIAM Journal on Computing,* 12 (1983), pp. 354–365.
19. Nassimi, D., and S. Sahni. "Data Broadcasting in SIMD Computers." *IEEE Transactions on Computers,* 20 (1981), pp. 101–106.

Where arę the parallel algorithms?

by ROBERT G. VOIGT
Institute for Computer Applications
in Science and Engineering
Hampton, Virginia

ABSTRACT

Four paradigms that can be useful in developing parallel algorithms are computational complexity analysis, changing the order of computation, asynchronous computation, and divide and conquer. Each is illustrated with an example from scientific computation, and it is shown that computational complexity must be used with great care or an inefficient algorithm may be selected.

INTRODUCTION

Parallelism has become a major contributor to increased performance in recent years, and it is now accepted that in the future supercomputers will involve many processors working together in parallel on a single problem. This trend has been brought about by fundamental limits on circuit switching and signal propagation times that inhibit dramatic increases in uniprocessor speeds, rather than by software developments that might exploit parallelism. In fact, the development of algorithms, programming languages, and operating systems is falling behind the pace of advances in hardware. We will focus on parallelism in algorithm development, emphasizing floating-point-intensive computations that arise in scientific computing.

There have been a number of parallel computers developed in recent years, but the majority of these have been of SIMD type; that is, a single instruction is applied simultaneously to a collection of operands. Even though these machines are architecturally different from vector machines, from an algorithmic point of view they are quite similar; it is natural to view an SIMD machine as executing instructions with vectors as operands.

Although some interesting research computers had been developed earlier at universities, we are just beginning to see machines of the MIMD type—where each processor can execute its own independent instruction stream in an asynchronous fashon. Unfortunately, few algorithms have been developed to take advantage of this potentially powerful form of computing. Algorithm development has been motivated by vector computers, and to a lesser extent by SIMD parallel arrays, because of their widespread availability.[1]

We will discuss four techniques that can be used as guidelines in the development of parallel algorithms. We will show in the next section that the first of these, computational complexity, must be used with great care to avoid selection of the wrong algorithm. Three other paradigms that can be useful in the development of parallel algorithms are changing the order of computation, computing asynchronously, and applying the divide-and-conquer concept. These will be illustrated in successive sections.

COMPUTATIONAL COMPLEXITY

Traditionally, one of the most important tools for evaluating algorithms has been computational complexity analysis; that is, operation counts. The fact that the fast Fourier transform of n samples requires $0(n \log n)$ arithmetic operations (here and throughout, log denotes ($\log_2$), whereas the straightforward approach requires $0(n^2)$, provides a clear choice of algorithms for serial computers. With the advent of vector computers—such as the Cray 1 and the Cyber 205, which have pipelined arithmetic units—computational complexity remains important because every operation costs some unit of time even if it is part of a vector operation. Thus, for vectors of length n, an algorithm that requires $\log n$ vector operations will not be faster for sufficiently large n than an algorithm that requires n scalar operations, because $n \log n$ operations must be performed. This preservation of arithmetic complexity was made precise by the concept of consistency.[2] An algorithm is said to be consistent if its arithmetic complexity is the same order of magnitude as that for the best serial algorithm.

Unfortunately, two important aspects of parallel computation are completely ignored by computational complexity and consistency. First, parallel computers can support extra computation at no extra cost if the computation can be organized properly. Second, parallel computers are subject to new overhead costs—required, for example, by communication and synchronization—that are not reflected by computational complexity. The importance of these concepts is illustrated by considering algorithms for the solution of the tridiagonal system of equations $Ax = b$.

If we consider an LU factorization of the matrix A where L is unit lower bidiagonal and U is upper bidiagonal, the usual algorithm is inherently serial. Defining the i^{th} row of these matrices as $(0,\ldots,0,c_i,a_i,b_i,0,\ldots,0)$, $(0,\ldots,0,l_i,1,0,\ldots,0)$, and $(0,\ldots,0,u_i,b_i,0,\ldots,0)$, respectively, the i^{th} element of the diagonal of U is given by

$$u_i = a_i - c_i \mathrm{b}_{i-1}/u_{i-1} \qquad (1)$$

and $l_i = c_i/u_{i-1}$. The solution x is obtained by solving $Ly = b$, followed by $Ux = y$, both of which require recursions similar to Expression 1. The computational complexity of the preferred algorithms is $0(n)$ for an $n \times n$ system.

Unfortunately, because u_i depends on u_{i-1}, Expression 1 and the other recurrences cannot be evaluated directly in parallel, and we are forced to consider alternatives. The most popular parallel algorithm is known as odd–even, or cyclic, reduction.[1] The idea is to eliminate the odd-numbered variables in the even-numbered equations by performing elementary row operations. Thus, if $R(2i)$ represents the $2i^{\text{th}}$ row of the tridiagonal matrix, the following operations can be performed in parallel for $i = 1,\ldots,(n-1)/2$, assuming n is odd:

$$R(2i) - (c_{2i}/a_{2i-1}) * R(2i-1) - (b_{2i}/a_{2i+1}) * R(2i+1) \qquad (2)$$

After the step indicated by Expression 2 is completed, a reordering again yields a tridiagonal system that is only half as large. Thus, in the case that $n = 2^k - 1$, the process may be continued for k steps until only one equation remains; then all of the unknowns are recovered in a back-substitution process. It has been shown that cyclic reduction requires $0(n)$ operations and is thus consistent.[2] Because the algorithm is consistent and because Expression 2 may be evaluated using vector operations, cyclic reduction has become the method of choice for vector computers. However, we will see that this may not be the case for parallel computers.

It has been noted that the elimination step (Expression 2) may be applied to every equation, not just the even ones, resulting in an algorithm known as odd–even elimination.[3] The equations are reordered and the elimination step is applied again. After k steps, for $n = 2^k - 1$, a diagonal matrix remains and the solution may be obtained in one more step without a back-substitution process. Because the elimination step is applied to every equation for $\log n$ steps, $0(n \log n)$ arithmetic operations are required. Thus,the algorithm is not consistent and is not a competitor on serial or vector computers. The situation is different on parallel computers. It is possible to organize the computation so that the extra work at each step does not require extra time; thus both odd–even reduction and odd–even elimination require $\log n$ steps. However, odd–even elimination requires no back-substitution phase, another $\log n$ steps for odd–even reduction, and has been shown to be superior on some parallel computers.[4,5]

Another potential advantage for odd–even elimination is that data movement potentially required by the back-substitution phase is unnecessary. This can reduce the communication requirements imposed by some parallel architectures.

Thus, good parallel algorithms can be ignored if one relies solely on computational complexity as a guideline. In particular, we must look for ways to perform extra computation in parallel if it will result in a reduction in the number of steps or in the amount of communication required.

ORDER OF COMPUTATION

The concept of changing the order of computation, reordering, may be viewed as restructuring the computational domain or the sequence of operations to increase the percentage of the computation that can be done in parallel. For example, in solving partial differential equations discretized over a grid, the order in which the nodes of the grid are numbered may increase or decrease the parallelism of the algorithm to be used. An analogous example is the reordering of the rows and columns of a matrix to create independent submatrices that can be processed in parallel. We have already seen how reordering a matrix can be beneficial; the odd–even reduction algorithm depends on reordering the equations between steps to preserve the parallelism.

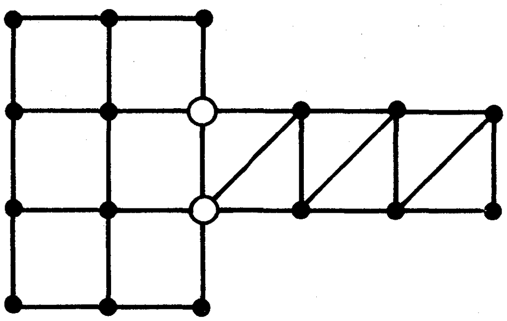


Figure 1—Domain of points

Selecting the order in which the points will be processed is a crucial step in solving a partial differential equation discretized on a grid of points representing the domain of interest. For example, if the points in Figure 1 are numbered left to right, top to bottom, the resulting linear system will have no particular structure other than being banded. It is important for serial computers to find orderings that reduce bandwidth because a smaller bandwidth means that fewer arithmetic operations are required to solve the system. Such other goals as numbering the points in order to increase the degree of parallelism may be more important on parallel computers.

Another method of ordering, known as substructuring, has been used to decouple structures that are connected by relatively few points.[6] This technique also can be used to introduce parallelism into the system. The situation is depicted in Figure 1, in which the circle points represent interface nodes between the two regions. The nodes in the region may be numbered in any appropriate order, but the interface points are numbered last. This gives rise to a block matrix of the form

$$\begin{bmatrix} A_1 & & C_1 \\ & A_2 & C_2 \\ D_1 & D_2 & B \end{bmatrix}$$

where the A matrices represent the two substructures, the B matrix represents the interface points, and the C and D matrices represent the dependencies between the interface nodes and the two regions. The A matrices may be factored in parallel, and then steps of the form $B - D_i A_i^{-1} C_i$ are used to eliminate the off-diagonal blocks. Finally, the modified B matrix is factored and the solution is obtained in a back-substitution process. This technique may be generalized to any number of substructures; the interface nodes must simply separate the structure. However, as the number of substructures increases, the size and complexity of the B matrix also increase, providing the algorithm designer with an interesting dilemma. This situation has been studied using a cube as a model.[7] Formulas were obtained to help in selecting the number of substructures so that the work involved in factoring the modified B matrix will not dominate all other computation.

Although this example involves the direct solution of the linear system, examples of reordering for parallelism exist for iterative methods as well.[1] The challenge is to find orderings that increase the degree of parallelism without increasing the arithmetic and communication complexity of some other aspect of the problem.

ASYNCHRONOUS COMPUTATION

Synchronization of computers of the MIMD type is used primarily in two situations. In the first, a value such as a sum must be computed from values in some subset of the processors before the computation can continue. In the second, synchronization is used to guarantee a specific order of computation that will reproduce the behavior of a traditional sequential algorithm.

Depending on the hardware and software of the system, synchronization may be time consuming. For example, several synchronization techniques were studied on the C.mmp computer system and the cost in terms of time used varied by a factor of 15, with some techniques requiring as much as 30 milliseconds.[8] Another, more subtle, cost of synchronization is poor processor utilization. Because all processors will not reach a synchronization point at the same time, those that arrive first will be idle until all are ready to proceed.

Thus we would like to consider algorithms that reduce the frequency of synchronization. Fortunately, there are situations in which the synchronization required by the computation of a value that depends on other values distributed throughout the system, or the synchronization required to mimic sequential behavior, may be eliminated by modifying the algorithm. The Jacobi iterative procedure for approximating the solution of a partial differential equation discretized on a grid requires computing a weighted average of values at neighboring grid points to update the approximation at a given grid point. A typical calculation is of the form

$$u_P^{k+1} = \frac{1}{4}\left(u_N^k + u_S^k + u_E^k + u_W^k\right) \tag{3}$$

for the north, south, east, and west neighbors of the point P. Clearly, this algorithm requires synchronization if the u values are being updated by individual processors in a parallel system. However, the algorithm may be modified so that Expression 3 is not forced to use values from the k^{th} iterate. This was the motivation for the pioneering work on chaotic relaxation[9] and for later studies.[10] In its simplest form chaotic or asynchronous iteration can be expressed as

$$u_P^{k+1} = \frac{1}{4}\left(u_N^{k-i_N+1} + u_S^{k-i_S+1} + u_E^{k-i_E+1} + u_W^{k-i_W+1}\right) \tag{4}$$

where i_N, i_S, i_E, and i_W are non-negative integers that may vary with k and P. In words, the algorithm suggested by Expression 4 would have each processor use whatever values were available to compute the next value of the iterate at a given point, regardless of which iterate those values were from. More sophisticated iterative schemes can be adapted to this form of computation.

The properties of asynchronous iterative methods are not well understood. Some theoretical work indicates that the methods will converge under conditions that guarantee the convergence of the corresponding sequential method if the values used on the right side of Expression 4 are from new iterates sufficiently often.[9, 10] Unfortunately, as with most iterative methods, the convergence results are asymptotic and do not provide much information about the observed rate of convergence. Some experimental studies on C.mmp that compare the performance of various asynchronous methods with sequential methods indicate that the asynchronous methods perform well.[10] However, the sequential methods chosen were not among the best available.

In addition to performance, two aspects of asynchronous iterative methods that require further study are convergence criteria and debugging techniques. Traditional convergence criteria require the computation of a value involving the sum of all solution approximations from the same iterate. At best, this requires periodic synchronization; at worst, it may not be possible because different approximations may be on dramatically different iterates. The difficulty with debugging asynchronous programs is that the values produced by the program may not be reproducible because the order of computation may change. This makes isolating errors very difficult.

DIVIDE AND CONQUER

The divide-and-conquer paradigm involves breaking a problem up into smaller subproblems that can be treated independently. Frequently, the degree of independence is a measure of the effectiveness of the algorithm because it determines the amount and frequency of communication and synchronization.

It is natural to apply the divide-and-conquer idea to the solution of differential equations by iterative methods. In Figure 1 the region could be divided between two processors with the squares in one and triangles in the other. An algorithm is executed independently in each processor, but information contained at the interface points indicated by the circles must be communicated periodically. Depending on the algorithm, synchronization may be required, but the techniques discussed in the previous section can be used so that the algorithms in the separate processors use whatever data are available, rather than waiting for synchronization. This idea can be extended to large two- and three-dimensional regions. One of the advantages the technique provides is that the region often can be subdivided to fit the number of processors.

Use of the divide-and-conquer paradigm also creates opportunities to balance the communication that is required among the various subpieces of the problem. For example, if the processor that is responsible for the square pieces of the region in Figure 1 updates values of the solution in a left-to-right, top-to-bottom order, then the values at the circled points will be available at different times. On some computer systems the communication of these values could be overlapped with computation. On the other hand, if the order of computation is top-to-bottom, left-to-right, then the values at the circled points will be available at essentially the same time, but not until the end of the computation. It would be much more difficult to overlap the required communication on most computer systems.

ACKNOWLEDGMENT

This research was supported under NASA Contracts NAS1-17070 and NAS1-17130 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Va. 23665.

REFERENCES

1. Ortega, J., and R. Voigt. "Solution of Partial Differential Equations on Vector and Parallel Computers." *SIAM Review,* June, 1985.
2. Lambiotte, J. Jr., and R. Voigt. "The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer." *ACM Transactions on Mathematical Software,* 1 (1975), pp. 308-329.
3. Heller, D. "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems." *SIAM Journal on Numerical Analysis,* 13 (1976), pp. 484–496.
4. Gannon, D., and J. Panetta. "Restructuring SIMPLE for the CHiP Architecture." *Parallel Computing,* 2 (1985).
5. Kapur, R., and J. Browne. "Techniques for Solving Block Tridiagonal Linear Systems on Reconfigurable Array Comuters." *SIAM Journal on Scientific and Statistical Computing,* 5 (1984), pp. 701–719.
6. Noor, A., H. Kamel, and R. Fulton. "Substructuring Techniques—Status and Projections." *Computers and Structures,* 8 (1978), pp. 621–632.
7. Adams, L., and R. Voigt. "A Methodology for Exploiting Parallelism in the Finite Element Process." In J. Kowalik (ed.), *High Speed Computation.* Berlin: Springer-Verlag, 1984.
8. Oleinick, P., and S. Fuller. "The Implementation of a Parallel Algorithm on C.mmp." Department of Computer Science Report No. CMU-CS-78-125, Carnegie-Mellon University, 1978.
9. Chazan, D., and W. Miranker. "Chaotic Relaxation." *Journal of Linear Algebra and Its Applications,* 2 (1969), pp. 199–222.
10. Baudet, G. "Asynchronous Iterative Methods for Multiprocessors." *Journal of ACM,* 25 (1978), pp. 226–244.

Panel: Future generation computers

Chair:
DHARMA P. AGRAWAL, *North Carolina State University,* Raleigh, North Carolina
Members:
PETER PATTON, *Computer Technology Corporation,* Austin, Texas
RAJ REDDY, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
HERBERT SCHORR, *IBM,* White Plains, New York
STEPHEN SQUIRES, *Defense Advanced Research Project,* Arlington, Virginia

Recent advances in VLSI technology and escalating demands have forced researchers to consider new ways to enhance processing capabilities. Researchers are involved with better understanding of the interaction and interdependency of various issues, including computer architecture, concurrent processing, knowledge representation, and artificial intelligence. This panel session provides an insight into various fifth- and future-generation computer projects. The pros and cons of employing alternative strategies are also emphasized.

Panel: VLSI arithmetic accelerators

Chair:
KAI HWANG, *University of Southern California,* Los Angeles, California
Members:
JOHN ELDON, *TRW—LSI Products,* San Diego, California
W. ERIC HALL, *Floating-Point Systems, Inc.,* Portland, Oregon
P. M. LU, *AT&T Bell Laboratories,* Holmdel, New Jersey
WILLIAM McALLISTER, *Hewlett-Packard Corporation,* Cupertino, California
BOB Y. WOO, *Weitek Corporation,* Sunnyvale, California

This panel session deals with state-of-the-art development of VLSI arithmetic chips. Panelists from industry concentrate on the consideration of 32-bit/64-bit floating-point arithmetic devices, vector/array processing, and scientific supercomputing.

Panel: Supercomputers and their impact on science and technology

Chair:
GEORGE PAUL, *IBM T. J. Watson Research Center,* Yorktown Heights, New York
Members:
JOHN W. D. CONNOLLY, *National Science Foundation,* Washington, D.C.
GEORGE DODD, *General Motors Research Laboratories,* Warren, Michigan
JAMES S. NOLEN, *J. S. Nolen and Associates,* Houston, Texas
VICTOR L. PETERSON, *NASA Ames Research Center,* Moffett Field, California
CARL SAVIT, *Western Geophysical Company,* Houston, Texas
LARRY SMARR, *National Center for Super Computer Applications,* Champaign, Illinois
JACK WORLTON, *Los Alamos National Laboratory,* Los Alamos, New Mexico

In the last few years there has been growing public awareness of the increasingly important role that supercomputers are playing in science and technology, as well as growing public concern regarding the diminishing lead of the United States in supercomputer technology. Government, industry, and academia have recently initiated cooperative programs to address supercomputing and to place it in its proper perspective as a national priority.

This panel examines the growing role of supercomputers in science and industry, its development over the last decade, and the future requirement for ultra-high performance systems and personnel properly trained to use them. Panelists also discuss current programs addressing these issues.

EDUCATIONAL AND SOCIETAL ISSUES

DAVID RINE, Track Chair
George Mason University
Fairfax, Virginia

Making Apple computers accessible to blind children

by SUSAN H. PHILLIPS, ANDREW G. RENOUF, and ROBERT A. BOWERS
Sensory Aids Foundation
Palo Alto, California

ABSTRACT

Sensory Aids Foundation engaged in a study examining computer-aided instruction (CAI) with 15 visually impaired students. Off-the-shelf educational software was adapted for use with a speech synthesizer. Students were tutored in typing, spelling, and language skills. The study was conducted for approximately 6 months, and the results were analyzed and are discussed in this paper.

A revolution has occurred that is dramatically changing the way we live. Even those of us with few or no technical skills are being affected by the advent of computers in our society. Today children are learning through computer-assisted instruction (CAI) and gaining computer literacy as early as preschool age. In 1983, Market Data Retrieval estimated that there were 200,000 computers in educational settings throughout the United States and that 600,000 could be expected by 1985. While many children are preparing for success in our computer world, blind and visually handicapped children have been denied this experience of CAI and computer literacy. If continued, their lack of experience will widen the gap between them and their sighted counterparts, needlessly compounding their handicap.

A review of the current literature indicates that the Apple computer has been made to "talk" with a limited amount of software. Phillip Schwartz, president of Computer System Resources in Florida, has adapted some software for the blind professional, as has David Holladay of the Raised Dot Computer Company, Madison, Wisconsin. In addition, Peter Maggs at the University of Illinois has developed software that enables the Apple to talk for specific uses by blind college students. There are also several voice synthesizers on the market that enable an Apple computer to speak whatever text is on the screen.

This work is primarily limited to use by visually impaired professionals and college students rather than by the general visually impaired populace or, specifically, visually impaired school children. Even the voice synthesizer peripherals now available for Apple computers are of limited use because of the lack of compatible software. Thus, it was with this backdrop of issues that the present project was undertaken. The project endeavored to test the feasibility of adapting off-the-shelf educational software to a speech synthesizer unit that was compatible with the Apple II, II+, and IIe personal computers and would be within the financial reach of most school districts. The project was to then make any necessary adaptations to the selected software and to test the efficacy of the speech synthesizer and adapted software in teaching children specific skills. The project was also concerned with the effect of CAI in motivating the participating children to do further computer work and how it might affect their feelings about computers in general.

To do this study, Sensory Aids Foundation received a grant from the United States Department of Education* for the period of August 1, 1983, to July 31, 1984. SAF, a nonprofit corporation, emphasizes the use of technological aids to assist visually or hearing impaired persons to achieve their greatest potential for independent and productive employment and education.

OBJECTIVES

Project objectives were as follows:

1. To develop the ability to use a variety of preprogrammed computer applications in an academic context. This included the ability to understand the purpose of, and discriminate between, the different software used.
2. To foster awareness by the students of the growing role of computers in our society and of their ability to function, vocationally or personally, in such a society.
3. To use the Apple II computer to develop special software that would provide access to off-the-shelf educational software. The Apple IIe was chosen for this project because there were more than 187,000 Apple computers in educational settings in the fall of 1983.
4. To use the Echo II speech synthesizer (because of its relatively low cost, high quality, and ease of installation).
5. To enable the students enrolled in the project to use the software developed by SAF. This includes the ability to use a keyboard and to run the software with supervision, but unaided.
6. To encourage the students in the project to consider vocational computer training just as they might consider other present vocational opportunities. Currently more than one million workers find employment in the computer industry, and this number will continue to increase.

SUBJECTS

The subjects for this project were 15 visually impaired students (7 boys and 8 girls) recruited from the San Francisco Bay Area. The children selected were in third through sixth grades. Because of the small number of visually impaired students in the area, and the fact that the study was a pilot project, criteria for selecting subjects were extremely flexible. However, extensive background data were collected to aid in interpreting the study data. This information included age, sex, degree and stability of vision, type of school program (mainstreamed or special education classes), reading level, and previous exposure to computers. The children had varying degrees of visual impairment but no other disabilities.

MATERIALS

The educational software packages used were a version of Master Type by Lightning Software (approval was received

* Grant #G00-83-00349

from Bruce Zweig prior to the sale of Lightning to Scarborough Systems), a spelling program, and a language arts program created by Sensory Aids Foundation. All three packages needed to be designed specifically for use with the Echo II.

PROCEDURE

All subjects were administered a battery of pre- and posttests: the Wepman Auditory Discrimination Test with the synthesizer speaking the words, a shortened version of the Minnesota Computer Literacy Questionnaire, a standard typing test (for the appropriate grade level) and the Stanford Achievement Tests for spelling and language (given at the appropriate grade level for each child).

Because of the large numbers of Apple II computers in the local school systems (due to the Apple Educational Program), the Apple II was chosen as the microcomputer for the study. It was felt that if the students in the study were to have further exposure to computers, it would probably be the Apple II. Therefore, using the same microcomputer would be consistent with probable future opportunities.

The speech synthesizer used in the study was the Echo II by Street Electronics. After a review of the available speech synthesizers, the Echo II was chosen because of its compatibility with the Apple II, its ease of use, and its cost, which was the least of all the synthesizers reviewed in October 1983.

Subjects were administered the pretests for auditory discrimination, computer literacy, keyboard proficiency, spelling, and language. The Project Director scheduled each student for 30 minutes per week of hands-on experience with the computer and voice synthesizer. Each student was taught to use both the hardware and the software. Basic information was given to each student on how a computer works. Students were allowed to explore the inside of the computer and actually feel the circuit boards. The students then received tutoring with an Apple II computer for 30 to 45 minutes each week for as many as 15 weeks. All 15 students received instruction with the Echotype program, 10 subjects received instruction with the spelling program, and 5 received instruction with the language program. At the end of the instruction period, posttests were administered.

ASSESSMENT

Because of the small number of subjects and the exploratory nature of the study, statistical analysis of data would not have yielded meaningful results. Thus, each case was looked at individually with the goal of identifying possible trends that would suggest further research questions.

RESULTS AND DISCUSSION

Because of the method of assessment and the nature of the results, the Results and Discussion sections will be combined so as to present the material more clearly. Thus, the project will be broken down by component and each will be assessed and discussed.

The Efficacy of the Speech Synthesizer

One of the concerns of the project was that the selected speech synthesizer, although possessing many other desirable attributes, would not produce speech sufficiently clear to be of use to students. This could be especially problematic with software such as the Echotype program, which required students to identify accurately single words—as opposed to words in a sentence, which can be better understood by using sentence structure and context as a guide. Our concerns about this matter were justified by the results of the Wepman Auditory Discrimination pretest given to the subjects. When two words (as spoken by the speech synthesizer) were the same, students correctly identified them as such 94% of the time within a range of 80% to 100%. However, when the two words were different, students responded correctly an average of only 44% of the time within a range of 33% to 53%. These scores were clearly due to the synthesizer's lack of fidelity, which made distinguishing similar-sounding words extremely difficult. Our hope was that with repeated exposure to the synthesizer, students would become accustomed to the rather mechanical speech produced by the synthesizer. Indeed, this proved to be the case, although not to as great an extent as we had expected. In the Wepman posttest, students averaged 97% correct responses to the same word pairs within a range of 80% to 100% and 56% correct responses to the different word pairs within a range of 47% to 70%. However, the students worked with the computer and speech synthesizer for a maximum of only 45 minutes each week. This small amount of exposure may have limited the extent to which the students could become adjusted to the synthesized speech. The experience of the two researchers who conducted the CAI, and who subsequently had a more prolonged exposure to the synthesized speech, was that with more exposure one becomes more accustomed to the speech. This finding was supported by the Wepman posttests, although the question remains to what extent students would adjust to the synthesized speech with more computer time. The difficulty of unclear speech can also be compensated for to some degree by adjustments in the actual software, which would change the pronunciations of words that are not consistent with the phonetic rules used in programming the synthesizer.

The Wepman posttest results, coupled with the clinical observations of the two researchers involved in monitoring the CAI, leads us to believe that the Echo II produces speech sufficiently clear for it to be effective in CAI. However, it is a recommendation of this project that when programmers are adapting software for use with the Echo II, one concern will be to insure that all the spoken text is pronounced correctly in order to facilitate students' understanding.

Software

The original intention of the project had been to take off-the-shelf educational software and to use it with little or no adaptation with the Echo II. We soon discovered, however, that almost without exception, educational software is very heavily graphics-oriented—many programs to the extent that they follow a video game format. Thus, software selection for the project became not just simply choosing appropriate software according to its instructional content, but also according to the format of that content. The only software that readily lent itself to adaptation was older, public-domain educational software, which was judged inadequate in terms of instructional content. Consequently, the Project Programmer made extensive revisions of the software finally selected. This was made possible by obtaining releases from the copyright holders for one piece of software so that the changes could be made. The remaining two programs were developed by SAF, because off-the-shelf software was unsuitable for adaptation.

Most software on the market today is protected so that a programmer may not enter the program in order to "look" at how it was written without knowing the entry code for the particular piece. The negotiating time needed to obtain permission from the software publishers to make the necessary adaptations is often six months or more. It would therefore be useful if writers of educational software could make available to special educators unprotected copies of their software.

Since this project was completed, the authors are now aware of two people actively engaged in reviewing public-domain software to be used with the Echo II speech synthesizer and therefore of use to blind people.*

CONCLUSION

As a result of this project, 15 visually impaired students in the San Francisco Bay Area have the same opportunity as their sighted peers to use a computer. Our society is moving with increasing speed into the computer era, and with the aid of projects like the present one, the visually impaired will be able to share in the new technology. Early exposure to computers is an important factor in motivating children to do further work with computers and to consider computer-related careers. For the visually impaired or blind child this also means another opportunity to participate in mainstream society.

As one student in the fourth grade said, "I feel more a part of the kids in my regular classroom because I can do the same stuff they can with a computer using the voice synthesizer."

SUGGESTED READINGS

1. "Educational Software." *Raised Dot Computing Newsletter,* 2 (1984), p. 12.
2. "Speech Access to Commercial Apple Programs." *Raised Dot Computing Newsletter,* 2 (1984), p. 21.

* Questions on the availability of the software should be directed to Susan Phillips, Sensory Aids Foundation, 399 Sherman Avenue, Palo Alto, California, 415/329-0430.

Teaching teachers to use microcomputers

by MARILYN D. WARD and PATRICIA L. HUTINGER
Western Illinois University
Macomb, Illinois

ABSTRACT

Project MUSE (Microcomputer Use in Special Education) is a federally funded training study housed in the College of Education at Western Illinois University. Its major goals are to develop and implement innovative techniques for training teachers to use microcomputers with handicapped children, to design curriculum strategies that reflect current technological advances, and to disseminate those materials to people who train teachers. In addition, the project is establishing a resource network for microcomputer software, hardware peripherals, developers, and users.

INTRODUCTION

Within the past five years there has been increased use of computers in the community and in schools. The U.S. is rapidly becoming a computer-dependent society, and computers affect almost every aspect of our lives. It is virtually impossible to use a telephone or a toaster, pay a bill, or travel on a train, plane, or bus without using a computer. There are microchips in such entertainment products as televisions and video games; in office products, such as word processors; in vending machines, elevators, and production line machinery. It is estimated that General Motors installed more than seven million microchips in automobiles last year.

Along with this proliferation of microcomputers has come a growing awareness that ordinary people can and must learn to use them. This has in turn created pressure for the introduction of computers to schools.[1] Groups ranging from educators to business leaders to parents have pushed for the placement of computers in elementary and secondary schools alike.[2] In the spring of 1982 estimates of the number of microcomputers in the schools ranged from 97,000 to 131,000.[2,3] Present estimates range from 325,000 to more than one million. Two-thirds of the nation's schools now have at least one microcomputer, and half have more than one.

In fact, the emergence of the microcomputer is changing and will continue to change the very fiber of American education. According to the Office of Technology Assessment:

> The so-called information revolution, driven by rapid advances in communication and computer technology, is profoundly affecting American education. It is changing the nature of what needs to be learned, who needs to learn it, who will provide it, and how it will be provided and paid for.[4]

B. F. Skinner states that computers can cure what's wrong with American education.[5] His premise is that computer-assisted teaching incorporates individual, interactive instruction that can give the student immediate positive feedback. Skinner believes it is this element of success that motivates students to learn. It is important to note that research findings appear to support his ideas. Several recent studies indicate that students learn more when they use computers than when they use traditional instructional materials. In a meta-analysis* of 51 studies of computer-assisted instruction, secondary students who used the technology scored better on objective tests than students who received traditional instruction only.[7] That same set of studies suggests that there may be an even greater effect on children in elementary schools.

As a result of these studies and others, and because of the general infusion of computers into our everyday lives and our schools, teachers are faced with a crisis. Many teachers, both preservice and inservice, find themselves ill-equipped to cope with the changes brought about by the new technology. Most teachers were educated in the industrial age, but now must function in the information age. This paper describes a project that is developing strategies to train teachers to use technology. It includes sections on the characteristics of teachers and trainers, the nature of training environments, and the content and competencies used in training.

PROJECT MUSE

Project MUSE (Microcomputer Use in Education) is funded by the Office of Special Education Programs, U.S. Department of Education, and housed in the College of Education at Western Illinois University. It addresses three major purposes. The first is to develop and implement an innovative university-based preservice and inservice special education curricular project to prepare personnel to use microcomputer hardware and software in the education of handicapped children and youth. The second purpose, related to the first, is to provide for the upgrading of professional knowledge and skills for practicing special educators and to update the course offerings for preservice special education personnel to reflect current technological advances. The third objective is implementation of the project's results in procedures and activities that are more effective for teachers and children alike than were elements of previously operating programs. Therefore, widespread dissemination of materials produced and procedures developed is part of the project's plan. In addition, the project is establishing a resource network for microcomputer software, hardware peripherals, developers, and users.

General strategies of the project include six major goals. These include the development and implementation of procedures to deliver coursework related to the use of microcomputers, software appropriate for curriculum to be used with handicapped children and youth, and techniques for teaching students to use both. Emphasis is placed on the use of existing, as well as modified versions of curricular software, for teaching the handicapped.

TEACHERS

The teachers who come to Project MUSE for training are a diverse group. They have different interests and come from different content areas (even when all are in special-educa-

* "Meta-analysis is a method of performing qualitative synthesis of diverse studies on a common topic."[6]

tion-related jobs). There also is a wide range in terms of computer sophistication. Although some participants have solid grounding in the use of computers and frequently are self-taught, others have never seen a computer. Some are intimidated by machines. A few teach computer courses or are responsible for computer training in their schools or agencies. Teachers may teach preschool or they may teach adolescents. Some work with severely handicapped individuals; others work with the mild-to-moderately handicapped. When possible, grouping is arranged by similar level of development or grade assignment.

Teachers are enthusiastic about learning to use computers. They come to sessions early and stay late, they volunteer materials, and they help each other. Frequently, they send ideas back to the project after they have started applying the results of their training.

TRAINERS

The best trainers not only have thorough knowledge of hardware, firmware, and software, but they have one absolutely essential ingredient, a sense of humor. They are reassuring, calm, and patient and are able to communicate on a level compatible with the teachers. They explain things in simple terms and answer questions at the level on which they are asked.

Computer experts have not met with much success in training teachers to use computers. Often they go too quickly and give too much detail. Experts frequently assume that beginners know as much as they do. They sometimes teach content out of sequence; they usually start with programming in BASIC. The MUSE project does not teach beginners programming with the exception of LOGO.

Good trainers function as members of a team; each has different skills and responsibilities. Although all trainers need to be competent with the use of hardware, firmware, and software, diverse skills relating to training adults also are important. Having an expert computer technologist on a training team is useful when working with more sophisticated teachers, but a person who is knowledgeable about handicapping conditions and educational programming for the handicapped is essential on a training team.

An understanding of adult learners' needs is critical to the success of a training team. Establishing a comfortable, nonthreatening environment is essential when training adults. Teachers who are learning to use computers frequently are apprehensive about the speed at which they learn and often worry about their ability to catch up. Some of the MUSE Project staff members have taken an intensive, two-week course designed to help them develop electronic devices, determine locations where the devices interface with computers, and master techniques for programming that use the logic of the computer with switches and an interface. After the experience of dealing with new terminology and learning to set up circuits with relays, potentiometers, and various timers, staff members are able to empathize with teachers who are fearful of computers and their accompanying unfamiliar vocabulary.

PHYSICAL ENVIRONMENT

Training environments are both physical and psychological. Physical necessities include proper equipment; adequate power supply including electrical outlets, heavy-duty power cords, and power strips; tables that place the computers at a comfortable height; availability of software, documentation, and related materials; tables and comfortable chairs where teachers can get away from the machines to read or write; a place for group discussions; and a place for snacks and informal interchanges.

Training sessions for beginners seem to work best when two people are paired on each computer. Even if there are enough machines for each person to work alone, it appears essential to have people work together initially. Pairing individuals with similar abilities works well. It is important to note that there should not be more than two persons to a machine. In that instance, one person seems not to have enough time on the computer.

Access to a large software library, with a wide range of programs, is necessary for effective training. Teachers must have the opportunity to look at content, determine program intent, and develop a sense of program capabilities.

Computer training, whether it takes place in a college course or an inservice course, should be scheduled to allow for large blocks of time. Inservice programs should begin with at least two full days with computers, followed by half-day sessions for review and acquisition of new skills. Formats that allow for a two- or four-week course, with blocks of four to five hours of scheduled class time in addition to open lab time, seem to work best. One hour a day after school once a week is not conducive to acquisition of computer skills. Administrative support for the project must be strong enough to allow teachers time for training during school hours.

CONTENT AND COMPETENCIES

Although the project's goal for special educators is to have them use computers in teaching handicapped children, MUSE Project training begins with computer literacy. The curriculum includes software use and evaluation, word processing, and use of LOGO and a wide range of peripherals. Training procedures are designed to help develop competencies that will make computer users out of teachers. Those teachers will have the skills and interest to use computer applications with children. Videotapes of applications in educating handicapped children are shown, but the focus is on helping teachers develop self-confidence with hardware, firmware, and software.

The MUSE Project has produced a number of written materials including microdictionaries, guides for machine use, software catalogs and evaluation forms, curriculum suggestions, lists of resource materials, and LOGO curriculum materials. A set of competencies for use of hardware, software, and peripherals also has been developed. The competencies covered range from beginning skills to more complex skills for higher level language programming and for interface and system design. They are arranged in sequence beginning with turning the computer on to diagnosing software problems, customizing programs, and interfacing noncomputer devices with computers. Equipment purchase is included, as is information on purchasing and evaluating software. Adaptation and modification requirements also are included.

CONCLUSION

The MUSE Project emphasizes a team approach to training. The need to establish trust and communication among a group of people with differing skills appears to be important in integrating computers into educational programs.

Training special educators to use hardware, firmware, and software is essential, but it is also important to recognize the human element. Teachers are still wary of computers, and trainers who attempt to start with complex programming skills, use jargon, present too much content in too few hours, fail to relate real situations to computer use, and fail to help teachers understand the full potential of the computer will only compound the problem. It is essential that we not underestimate the negative effects of this wariness and fear because it can prevent the use of the computer in education of handicapped children as effectively as the lack of electricity would. Teachers must be eased comfortably, carefully, and with empathy from the industrial age into the information age.

ACKNOWLEDGMENT

This work was developed with funds provided by a grant from the Office of Special Education Programs, Department of Education. The content of this paper does not reflect the position or policy of the U.S. Department of Education, and no official endorsement of this paper should be inferred.

REFERENCES

1. Zemke, S. "Microcomputers and Education—Choices and Consequences." *Educational Computer Magazine,* (March) 1983, pp. 42–45.
2. Ingersoll, G., and Carl Smith. "Availability and Growth of Microcomputers in American Schools." *T.H.E. Journal,* 12-1 (1984), pp. 84–87.
3. Grayson, L. "An Overview of Computers in U.S. Education." *T.H.E. Journal,* 12-1 (1984), pp. 78–83.
4. Shane, H. "The Silicon Age II: Living and Learning in An Information Epoch." *Phi Delta Kappan,* (October) 1983, pp. 126–129.
5. Zientara, P. "B. F. Skinner: Computers Can Cure What's Wrong in American Education." *Infoworld,* (February 6) 1984, pp. 23–25.
6. Slavin, Robert. "Meta-Analysis in Education: How Has It Been Used?" *Educational Researcher,* 13, pp. 6–15.
7. Barbour, A. "Computing in America's Classrooms 1984." *Electronic Learning,* 4-2 (1984), pp. 39–44, 100.

Computer contributions legislation and the public interest

by HAL BERGHEL
University of Nebraska
Lincoln, Nebraska

ABSTRACT

This paper critically examines a type of legislation currently pending before both houses of Congress. This legislation seeks to encourage computer manufacturers to donate computer equipment to school systems by giving them preferential treatment under the tax code. It is argued that this sort of legislation is not in the public interest.

INTRODUCTION

In a recent exchange with Congressman Pete Stark,[1] I argued that the computer contributions legislation pending before both houses of Congress was not in the public interest. Regrettably, some of the points raised in this article were misunderstood. It is my intention to clarify these points in this paper as well as expand upon some related issues which were only briefly dealt with in the prior article.

BACKGROUND

I shall use the phrase "computer contributions legislation" to refer to any type of legislation intended to encourage the donation of computer equipment to schools by means of modifications of the Internal Revenue Code (IRC) and provide computer manufacturers with a special tax advantage not otherwise available to other manufacturers.

The legislation discussed in the original articles[2–8] all recommended the same modification: subsume computer contributions under a so-called *special rule* and effectively allow the contributor to deduct twice his basic investment in the property from his federal tax liability. Since nearly all computer manufacturers are in the highest (46%) corporate tax bracket, this legislation would typically result in a tax deduction equal to 92% of the cost of production of the contributed property. By any standard, a contribution with a reward of this magnitude stretches the term *charitable*. Perhaps a more descriptive term would be *painless*.

I object to this sort of legislation for a variety of different reasons and at a number of different levels. For one thing, it is inherently unfair. It singles out a particular segment of the economy for special treatment under the tax laws. In my view, this is only slightly less objectionable than conferring privileged status to individuals or a select group of corporations. The basic spirit of this legislation is anti-egalitarian, and should be opposed for that reason alone.

This is not the only theoretical objection. In addition, this legislation conceals the expense of the program. Unlike appropriations legislation, where the public cost is prescribed in the legislation itself, the cost of this legislation can only be determined after it has been incurred. It seems to me that this is tantamount to trying to budget money after it has been spent. When the true cost of the legislation is known, it will be too late to do anything about it.

Further, most of the bills fail to restrict either the amount of corporate contribution or the amount of deduction. Incredibly, such decisions are left to the discretion of the contributor. Placing the cost controls of a public program in the hands of those who stand to reap the greatest rewards is not a sound policy. Even the more conservative bills allow individual contributions of up to 20% of a corporation's annual sales. We may be dealing with some staggering figures.

I do not oppose this legislation on purely theoretical grounds. There are also myriad practical objections that can be raised. For one thing, this legislation will dramatically reduce the cost of overproduction. It allows the manufacturer who overproduces his product, that he stands to lose at most 8% on his capital investment and recover the remaining 92% through reductions in his federal tax liability by finding some school to accept the unsold units as a gift. I suspect that such schools would not be terribly difficult to find. This type of legislation can in essence buffer computer manufacturers from the harsh consequences of unsound production decisions. Rewarding corporations for strategic miscalculations of this sort is clearly inconsistent with the principle of free enterprise.

My reservations concerning this legislation are not limited to the overall tax implications. Another practical concern is that these bills are, as a whole, very poorly thought out. As an illustration, consider the *allowed contributions* section of Table I. Note that three of the bills specifically exclude software from consideration. It seems both arbitrary and unreasonable to accord only hardware manufacturers this privileged status. Without the software, the hardware is inert. If our objective is to put these computers to work in the schools, we need the software as much as the hardware.

In addition, most of the bills specifically exclude such services as installation, routine maintenance and repair. As any owner of a microcomputer will testify, the expense of such support services is considerable. These machines will fail, and when they do it is unlikely that the teachers and students will have the expertise to repair them. A school lacking the resources to maintain this type of acquisition may find it to be an onerous gratuity.

Perhaps the strangest features of all are those which refer to hardware requirements. These specifications seem to have been handled in a most cavalier fashion. For example, while six of the seven bills require that the computer support three programming languages, four of the seven only require 16K of primary memory. No compiler or CAI package worthy of the name, however, can work within such a restrictive environment. These requirements are inconsistent with the realities of the educational use of computers.

Further, three of the bills do not require that the computer be equipped with a monitor. Six of the seven do not even require secondary storage and four of the seven do not require that the computer be suitable for educational use. Exactly what kind of equipment are we soliciting here?

If the intent is to put the computers to work in the schools,

Table I—Major features of pending computer contributions legislation

	H.R. 91	H.R. 701	S. 108	H.R. 2417	S. 1194	S. 1195	H.R. 3098
Date of Introduction	1-3-83	1-6-83	1-26-83	5-3-83	5-3-83	5-3-83	5-23-83
IRC	170(e)(1)	170(e)(1)	170(e)(1)	170(e)(1)	170(e)(4)	170(e)(4)	170(e)(4)
Duration	1 yr	1984	open	1 yr	5 yrs	5 yrs	5 yrs
Focus							
Public Schools	X	X		X	X	X	X
Low/Mid Inc. Public Schools	X[1]						
Vocational			X			X	
Higher Ed.					X	X	X
Museums					X		X
Prisons					X		X
Allowed Contributions							
Hardware	X	X	X	X	X	X	X
Software				X	X	X	X
Noncomputer Instruments					X	X	X
Services					X	X	X
Max. Corporate Contribution	open	open	open	open	20% of annual sales		
Max. Corporate Deduction	open	open	open	open	10% of annual taxable income		
Restrictions on Contributor							
-must be manufacturer	Y	Y	N	Y	Y	Y	Y
-max. age of equipment[3]	6 mos	6 mos	open	6 mos	6 mos	6 mos	6 mos
-unused	Y	Y	open	Y	Y	Y	Y
-100% contribution	Y	Y	Y	Y	Y	Y	Y
-plan[2]	N/R	N/R	N/R	Y	Y	Y	Y
Max. Deduction/Unit (% of basis)							
Hardware	200%	200%	200%	125%	200%	200%	200%
Software	N/A	N/A	N/A	N/A	FMV	FMV	FMV
Hardware Requirements							
Languages Supported	3	3	open	3	3	3	3
Min. Primary Storage	32K	32K	open	16K[4]	16K[4]	16K[4]	16K[4]
Monitor Required	No	No	No	Yes	Yes	Yes	Yes
Peripherals Required	none	none	none	DISK only	none	none	none
Must be Suitable for Educ. Use	N/R	N/R	N/R	Y	Y	N/R	Y
Must be Covered by Warranty	N/R	N/R	N/R	Y	Y	Y	Y
Restrictions on Use							
-primarily for student education	Y	Y	Y	Y	Y	Y	Y
-compliance guarantee	Y	Y	Y	Y	Y	Y	Y
-governing body consent	Y	N	N	Y	Y	Y	Y

LEGEND: N/A = not allowed N/R = not required FMV = fair market value

[1] At least 75% of total contribution must be to low/middle income schools.

[2] Four bills require that contribution be made pursuant to written plan guaranteeing equitable distributions of property.

[3] Since date of manufacture.

[4] Must be expandable to 48K.

From Berghel.[1] Copyright 1984 by Association for Computing Machinery. Reprinted with permission.

we need the monitors, peripherals and software just as much as the mother board and vinyl case. Despite this need, six of the seven bills require that the donations be restricted to computer manufacturers while manufacturers of peripherals and software houses are specifically excluded. This simply makes no sense.

In summary, I do not believe that it is in the public interest to manipulate the tax laws for the benefit of a particular industry. However, even if this objection is set aside, these particular pieces of legislation are unworthy of consideration, for they contain provisions which are inconsistent with their own avowed objectives. These bills should be seen for what they are: self-serving pieces of legislation motivated by hardware manufacturers for hardware manufacturers.

Perhaps the most upsetting aspect of this legislation is that the school systems are taken in as unwitting accomplices. School systems are always faced with a shortage of funds. It is simply unreasonable to expect them to turn away free equipment, even if they have no immediate need. A byproduct of this legislation is that it encourages school systems to behave in a manner which, if generalized, is fiscally irresponsible. The schools are encouraged to accept all of the free computer equipment that they can find and the cost will not be reflected in their budget. Further, the federal taxpayer will not be aware of the costs involved until after the fact.

THE PURPOSE OF THE LEGISLATION

The intent of this legislation is to infuse large numbers of microcomputers into the public school systems. Typically, proponents of the legislation[9, 10] argue that this is necessary for two reasons: (1) because computers facilitate the teaching process, and (2) because computer literacy will be a prerequisite for jobs of the future. For the sake of convenience, we shall call these the *pedagogical* and the *computer literacy* hypotheses, respectively.

The pedagogical hypothesis is certainly appealing. We are continuously reminded of the virtues of computer assisted instruction through professional journals, monographs, anthologies and conferences. We are told of the long-term economies of CAI, its effectiveness and intellectually challenging nature, and of its ability to adapt itself to the level of the student. And, of course, there are psychological advantages for students who require a great deal of positive reinforcement or are too timid or recalcitrant to participate fully in the classroom experience. The arguments in favor of using CAI in the schools are quite compelling.

But what does the pedagogical hypothesis logically entail? Suppose for the moment that we grant that computers have considerable potential in the classroom. Does this justify the unrestricted acquisition of computers by the schools? Absolutely not! But that is the point of the legislation under consideration.

The fact that computers have all of the alleged educational advantages no more justifies this type of surreptitious tax increase than the realization that books are the mainstay of education justifies socializing the printing industry. Computers are educational tools in the same sense that chalkboards, overhead projectors, televisions and video cassette recorders are educational tools. When they are used in appropriate contexts by suitably trained personnel, they can be of enormous value. However, it will be primarily the context and the staff which will determine their effectiveness. It is not the scalpel which makes for successful surgery, but the surgeon using the scalpel.

So even if the pedagogical hypothesis is correct, it doesn't support the kind of legislation which has been proposed. The course of action which it implies is that we should continue to devote resources to CAI so that we can ultimately take full advantage of this new educational instrument.

THE COMPUTER LITERACY HYPOTHESIS

Let's examine, for a moment, the hypothesis which deals with computer literacy. This hypothesis claims that computer-related, or at least high-tech, jobs will predominate in the future. The supportive argument usually runs something like this: we are now in the early stages of what Bell[11] and others have called the information age; most jobs in the information age will require computer literacy; thus, if the current students are to be gainfully employed in the future, they must be computer literate. Even if this hypothesis is correct it does not follow that computer contributions legislation is the best solution. However, if we are indeed in the throes of a crisis, we can certainly tolerate otherwise unacceptable excesses. But are we in the throes of a crisis?

Arthur Luehrmann, Director of Computer Research at the University of California at Berkeley, thinks that we are. In his words, "Computing plays such a crucial role in everyday life and in the technological future of this nation that the general public's ignorance of the subject constitutes a national crisis."[12] This sentiment is echoed in a recent publication from the Education Commission of the States.[13] Consider the following passage:

> Occupational growth throughout the 1980s is projected to expand most rapidly in the higher-skilled, technical occupations. Tomorrow's workers will likely need improved skills in the selection and communication of information. Many of today's skills considered to be of a "higher" level are the potential basic skills of tomorrow.

So apparently there is a widespread belief in the computer literacy hypothesis. And if this hypothesis is correct, we should take immediate action.

The proponents of the computer literacy hypothesis would have us believe that the jobs of the future will favor those with professional/technical backgrounds, and even the traditional jobs will require ever-increasing skill levels. Let's take up these two in turn.

Will the preponderance of future jobs require a professional or technical background? The evidence seems to point in the opposite direction. As an illustration, consider some recent projections from the Bureau of Labor Statistics. It estimates that between 1978 and 1990 there will be 200,000 new jobs for computer systems analysts, an increase of over 100%. During this same time, however, there will be over

600,000 new jobs for janitors and sextons, and over 800,000 new jobs for fast food workers.[14] While the rate of employment growth in high-tech areas is faster (it is estimated that they will increase by 45% in this decade[15]), their employment base is so small that they will contribute only a small fraction of the total number of jobs available by the end of the decade.

Perhaps the most thorough research in this area is currently being conducted at Stanford.[16–22] Let me summarize a few of their findings:

1. Of the 20 occupations expected to generate the most jobs from 1978–1990, not one is related to high-tech.
2. Only three or four of these occupations require education beyond high school.
3. BLS estimates show that high-tech occupations will account for only 7% of all new jobs during the 1980's, and only 4% of the new jobs during the period 1982–1995.
4. It is expected that from 1978–1990 there will be more than five times as many new jobs created for fast food workers and kitchen help as for computer programmers.
5. The five occupations which are expected to generate the most new jobs require low skill levels. They are: janitors, nurses' aides, sales clerks, cashiers and waiters.

Contrary to what some might wish to think, the evidence seems to indicate that the lion's share of new jobs in the 1980's and 1990's will be low-tech rather than high-tech.

Similar conclusions are reached when one considers the effects of automation on the skill levels of extant jobs. The popular misconception is that technological advances, particularly those associated with computers, will require greater skills from future workers. Indeed there have been studies which have shown that automation has increased requisite skill levels.[22–25] However, there are also studies which showed the converse.[26–29] Not surprisingly, the correct view seems to be that automation has varying effects on skill level at different times and in different industries.[30–32] When we ignore the impact of automation on skill level on particular industries and look at overall trends, we see that skill levels have remained relatively constant in the past few decades despite automation.[16–18]

Let me conclude this section with a quotation from Rumberger and Levin,[21] the Stanford researchers mentioned above:

> The message . . . is clear: not only will high tech provide few job opportunities in the future economy, but most new jobs will require no post secondary schooling and will pay wages lower than average. . . . According to the BLS projections, high tech will not dominate the future job market. . . . Instead, future job growth will favor service and clerical jobs that require little or no post secondary schooling and pay below-average wages.

TECHNOLOGY, TEACHING, AND TAXES

As I mentioned above, enthusiastic support for computer contributions legislation usually results from belief in at least one of two hypotheses. I have argued that even if the pedagogical hypothesis is correct, it certainly doesn't justify anything as drastic as a distortion of the tax laws to benefit one industry. On the other hand, the computer literacy hypothesis is most likely false. Why, then, are we seriously considering this legislation?

The answer lies in human nature. We live in an age of computer mania. High-technology is a boom industry. Computer people earn substantial salaries. Computer sales seem to increase every year even when the economy is sluggish. What could be more natural than to want our children to get a piece of the action. And even if our children aren't directly involved in the computer field, we certainly don't want them left out in the cold when their chosen occupation is automated. Moreover, the contributed computers don't actually cost anything. They are donated by the manufacturer.

Even though this view is understandable, it is based on several misconceptions and motivated by misplaced fears resulting from a serious distortion of the facts. The computers are not free and current evidence indicates that computer related skills will not dominate work settings of the future. Even though the educators and legislators who hold this view are well-intentioned, they are reacting to an imagined problem rather than a real one.

The idea of increasing the use of computers in the schools, in a purposeful and productive fashion, is sound. There is no question about that. We should, however, not lose sight of the fact that our real objective is the increase in overall *quality* of the educational experience and not the *quantity* of instructional aids. I suggest that the solution to any computer literacy problem lies in a return to the concept of a diversified, well-rounded education rather than in the acquisition of more hardware. If our children leave the schools with excellent communication skills and a firm background in mathematics and the physical and social sciences, the future will hold great promise. However, in the absence of such training, no degree of exposure to computers will overcome this deficiency.

LEGISLATIVE UPDATE

At this writing, none of the bills discussed have passed through Congress. However, at least nine additional pieces of legislation have been introduced. Eight of these nine bills call for direct appropriations to the states and/or educational agencies for the support of computer literacy programs. One bill seeks to modify the IRS code, but for the purpose of encouraging teacher training in computers. No new legislation has been introduced which will propose modifications in the tax code and benefit computer manufacturers.

At the state level, ten states (California, Illinois, Indiana, Kansas, Louisiana, Maine, Maryland, Montana, Pennsylvania and Rhode Island) have proposed computer contributions legislation. Of these, four (California, Indiana, Louisiana and Maine) have passed them into law. Thus, this issue is still very much alive at both the state and federal level.

Appropriations legislation has been considered in Arkansas, Connecticut, Florida, Illinois, Kentucky, Minnesota, New Jersey, New York and Pennsylvania, and has been enacted in all but New Jersey and Pennsylvania.

REFERENCES

1. Berghel, H. "Tax Incentives for Computer Donors is a Bad Idea." *Communications of the ACM, 27* (1984), pp. 188-193.
2. S. 1194. "Technology Education Assistance and Development Act of 1983." 98th Congress, 1st session (1983).
3. H.R. 91. "Computer Equipment Contribution Act of 1983." 98th Congress, 1st session (1983).
4. S. 108. Untitled. 98th Congress, 1st session (1983).
5. H.R. 3098. "Technology Education Assistance and Development Act of 1983." 98th Congress, 1st session (1983).
6. H.R. 3750. "Computer Literacy Act of 1983." 98th Congress, 1st session (1983).
7. H.R. 2417. "Computer Contribution and Teacher Training Act of 1983." 98th Congress, 1st session (1983).
8. H.R. 701. "Computer Contribution Act of 1983." 98th Congress, 1st session (1983).
9. Brown, J. "Computers and the Schools." *Technological Horizons in Education, 10* (1982), pp. 99–100.
10. Stark, F. "The Best Way to Put Computers into the Schools Today." *Communications of the ACM, 27* (1984), pp. 186–194.
11. Bell, D. *The Coming of a Post-Industrial Society.* New York: Basic Books, 1973.
12. Luehrmann, A. "Computer Illiteracy—A National Crisis and a Solution for It." *Byte,* 5 (1980), pp. 98–102.
13. Education Commission of the States (Report). "The Information Society: Are High School Graduates Ready?" Denver: Education Commission of the States, 1982.
14. Carey, M. "Occupational Employment Growth Through 1990." *Monthly Labor Review, 104* (1981), pp. 42–55.
15. Coleman, G. Memorandum on BLS employment estimates, unpublished manuscript, 1982.
16. Rumberger, R. "The Changing Skill Requirements of Jobs in the U.S. Economy." *Industrial and Labor Relations Review, 34* (1981), pp. 578–590.
17. Levin, H., and R. Rumberger. "The Future Impact of Technology on Work Skills." In H. Didsbury, Jr. (ed.), *The World of Work.* Bethesda: World Future Society, 1983.
18. Rumberger, R. "The Job Market for College Graduates 1960–1990." Project Report 83-A3, Institute for Research on Educational Finance and Governance, Stanford University, February 1983.
19. Levin, H., and R. Rumberger. "The Educational Implications of High Technology." Project Report 83-A4, Institute for Research on Educational Finance and Governance, Stanford University, February 1983.
20. Rumberger, R., and H. Levin. "Forecasting the Impact of New Technologies on the Future Job Market." Project Report 84-A4, Institute for Research on Educational Finance and Governance, Stanford University, February 1984.
21. Rumberger, R. "High Technology and Job Loss." Project Report 84-A12, Institute for Research on Educational Finance and Governance, Stanford University, May 1984.
22. Helfgott, R. "EDP and the Office Work Force." *Industrial Labor Relations Review, 19* (1966).
23. Jaffe, A., and J. Froomkin. *Technology and Jobs: Automation in Perspective.* New York: Praeger, 1968.
24. Eckaus, R. "The Economic Criteria for Education and Training." *Review of Economics and Statistics, 46* (1964), pp. 181–190.
25. Evans, J. "The Worker and the Workplace." In G. Friedrichs and A. Schaff (eds.), *Microelectronics and Society.* New York: Mentor, 1982.
26. Evans, J., op cit.
27. Zimbalist, A. "Technology and the Labor Process in the Printing Industry." In A. Zimbalist (ed.), *Case Studies in the Printing Industry.* New York: Monthly Review Press, 1979.
28. Bright, J. "Does Automation Raise Skill Level Requirements?" *Harvard Business Review,* July (1958), pp. 85–98.
29. Braverman, H. *Labor and Monopoly Capital.* New York: Monthly Review Press, 1974.
30. Crossman, E., and S. Laner. "The Impact of Technological Change on Manpower and Skill Demand: Case Study and Policy Implementations." Department of Industrial Engineering and Operations Research Report, The University of California—Berkeley, 1969.
31. Gotlieb, C., and A. Borodin. *Social Issues in Computing.* New York: Academic Press, 1973.
32. Mowshowitz, A. *The Conquest of Will: Information Processing In Human Affairs.* Reading, Mass.: Addison-Wesley, 1976.

Programming environments based on structure editing: The GNOME approach

by RAVINDER CHANDHOK, DAVID GARLAN, DENNIS GOLDENSON, PHILIP MILLER and MARK TUCKER

Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

The use of integrated programming environments based on structure editing is an emerging technology that has now reached the stage of being both demonstrably useful and readily implementable. We have outlined some of the salient aspects of our work in developing the GNOME and MacGNOME programming environments and suggested paths of implementation that seem to be worth traveling. A predominant theme in all of this has been the need to separate policy from mechanism. While the choice of user interface policies will probably differ widely from those we have made here, the mechanisms that we have sketched will nonetheless be applicable to future environments.

INTRODUCTION

Traditionally, program development has taken place in what remains a very poorly integrated environment. A program typically is constructed using a text editor, then translated in a stand-alone compiler, while error correcting is done with the aid of separate debugging tools. Each piece of software, and the host operating system, has its own unique command language. Hence, a good deal of expert knowledge is required to use the tools effectively. This leaves much to be desired from the point of view of both expert and novice programmers. The novice must focus early attention on extraneous details of tool invocation at the expense of concentration on abstraction and problem solving. The expert probably knows a good deal about such details, but these details still can be time consuming and error prone.

There has been considerable discussion recently about the development and potential of structure editing to alleviate this problem by supporting *integrated programming environments.*[1–6] These environments can be viewed as a collection of cooperating tools that communicate to the user through a single uniform interface combining the functions of a text editor, interpreter, compiler, and debugger. Our own work has focused on the development of integrated programming environments designed for the teaching of block structured, high level languages to novice users.

Many computer science educators agree that it is important that the beginning programmer meet with early, easy, and immediate success. Simple programs should be simple to write. They should execute rather than respond with cryptic messages about parsing failures or declaration shortcomings. Additionally, the first few days and weeks of programming are times when programming habits are formed. If the student's attention is focused on the location of semicolons and the spelling of predefined types, it isn't focused on problem-solving, top-down design, modularization, and abstraction.

When constructing a program in a traditional text editor, one enters a program as if it were no different from any other textual document. The editor edits characters that just as easily could be part of a business letter or poem. Hence, it is easy to make punctuation and other syntax errors, errors which often are difficult for a novice to detect. Structure editors, on the other hand, are syntax-directed. They are knowledgeable about the programming language being used. Rather than single characters, their basic elements are syntactically meaningful entities. A syntax-directed editor for a procedural programming language, for example, operates on data declarations, subprogramming units, statements, expressions, and so on. The editor allows no constructions which are not syntactically valid; since it is then impossible to make a syntax error, the novice programmer is freed to concentrate on higher levels of program abstraction.

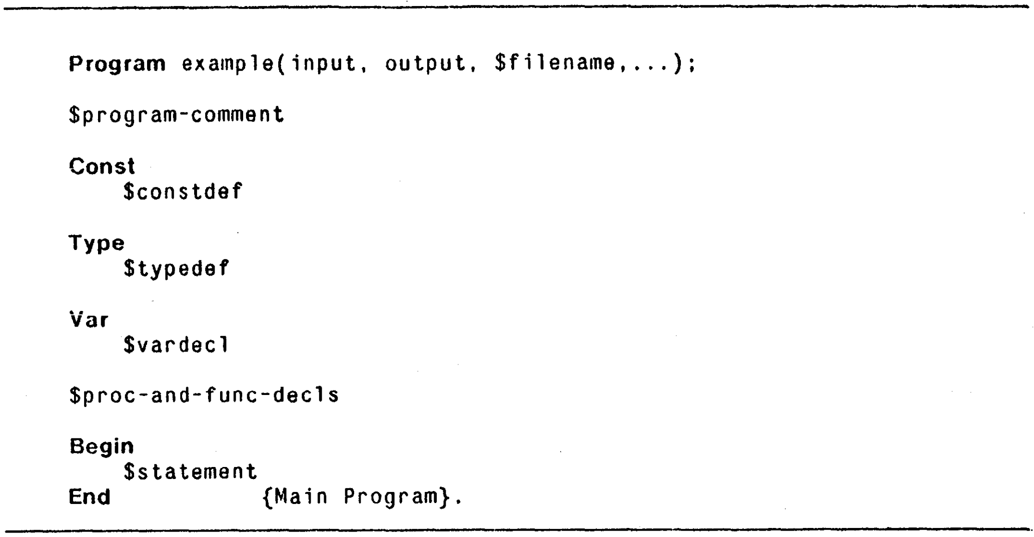
While the structure editor prevents syntax errors, other tools in the environment combine to support the various activities concerned with producing a correct and runnable program. In particular, an incremental semantic analyzer can check for "compile-time" correctness as the program is entered; some semantic errors can be prevented altogether, others can be reported to the programmer in the immediate context in which they are made. Run-time tools can allow a mix of editing, compiling, interpreting, and running supporting various active views of the program. Other tools provide an interactive on-line help system, smooth interface to the file system, and documentation aids.

The remainder of this paper describes some of the work we have done in developing structure editing environments at Carnegie-Mellon University. It highlights several aspects of the work that we hope will be of interest from the ponts of view of both users and implementors, computer science educators, and programming systems professionals. The first two sections give some background about the GNOME environments on which we have been working, what they look like to a user, how they work, and why we think they provide a useful alternative to traditional approaches to programming.

When designing software, it is useful to separate design issues involving system functionality from those which are strictly a matter of user interface. We characterize this distinction as one between *mechanism* and *policy.* In describing some of the important lessons we have learned in implementing the GNOME environments, the predominant theme is that by clearly distinguishing between mechanism and policy, an implementor can provide a wide variety of policies using relatively easily implemented mechanisms. This is done by shifting much of the burden of maintaining an incremental environment off the shoulders of elaborate data structures and onto the shoulders of efficient procedures that can regenerate information as needed.

GNOME PROGRAMMING ENVIRONMENTS

Many syntax-directed environments work by presenting the programmer with a program template directing attention to points where further program construction is permissible. For example, in the GNOME environment, a program initially looks as depicted in Figure 1. The words prefixed with a $ symbol are at those points in the program where further construction is valid. They are called *metanodes,* and correspond to the non-terminals of a formal specification of the pro-



```
Program example(input, output, $filename,...);

$program-comment

Const
    $constdef

Type
    $typedef

Var
    $vardecl

$proc-and-func-decls

Begin
    $statement
End            {Main Program}.
```

Figure 1—A GNOME program skeleton

gramming language.* The editor automatically displays all syntactic "sugar" (keywords, punctuation, etc.) and formats the program. The current focus of attention, or *cursor,* is indicated in GNOME environments by a boxed area that is displayed with inverse video. Allowable choices for construction at any metanode are displayed in a separate help *window.* The programmer need only uniquely identify the desired construction and it is inserted into the program tree.

GNOME maintains the program internally as a tree. Rather than parsing text into a tree that is then compiled into machine code, GNOME grows and shrinks the program tree, *unparsing* the tree representation into a text representation that the programmer sees at the terminal.

Until recently, structure editors have not been widely available for the broad spectrum of classrooms in which programming is taught, nor for serious programming outside of a few well-supported universities and research institutes. Some existing structure editor environments work only on subsets of languages, and handle only small programs. Others require expensive time-shared systems run on exotic operating systems, or depend on expensive terminals. Still others use prohibitively costly personal computers.

MacGNOME is an implementation of a GNOME environment targeted for the Apple Macintosh computer. Through MacGNOME, we intend to make structure editor environments widely available, taking advantage of the high resolution graphics capability and 32-bit architecture found in the emerging crop of affordable personal computers. Initially, we are working on two language environments, Pascal[7] and Rich Pattis' *Karel the Robot.*[8] Use of the Macintosh ensures that the environment will be readily available, and of sufficiently low cost to allow its use by secondary schools as well as universities. The environment supports full ISO Standard Pascal programs of moderate size and provides all of the kinds of integrated program development support outlined above.

MacGNOME ENVIRONMENTS: THE USER'S PERSPECTIVE

In this section, we sketch how the user interacts with a MacGNOME environment by focussing on four important aspects of the system, the views that the user has of a developing program, the way the user navigates around a program, how he constructs a program, and how errors are reported to him.

Program Views

Both text and structure editors tend to impose a single view of the program. Text editors present a program as a linear stream of characters, in full detail. In the style of Pecan and Gandalf, MacGNOME allows multiple "views" of a program.[2,5,9] These provide alternative visualizations of a program, appearing in multiple windows on the screen.

MacGNOME supports both a linear text and a "scoped" view of the program. In the former instance, the bodies of subprograms or other large blocks of code can be collapsed into icons and hidden from view, or left expanded in place. The latter view displays the program using separate windows for each subprogram. This means that only the subprogram headers are visible in the blocks where they are declared, but both the header and its contents are visible in a separate window when the programmer wishes to access them.

We are also implementing an outline view of the program. It focuses the programmer's attention on scoping and nesting issues. This view is particularly helpful for organizing a program at high levels of abstraction, and for navigating through the program when it gets large and detailed. Indeed, some prefer to construct the program initially using this view, then moving into a scoped view to fill out the details.

Within a particular view of a program, it is also possible to affect the display of various components of the grammar by not displaying certain constructs of a language in the default

* Our use of comments is something of an exception to a direct correspondence.

presentation. This has been particularly helpful in educational settings where it is used to hide constructs of the Pascal grammar with which students are not yet familiar.

Moving Through the Program

With a traditional text editor, default cursor movement is not an issue; one types text sequentially until it is time to change what was initially written. However, with a structure editor, the basic entities are meaningful in terms of the language being edited. So, decisions of style and method must be made about the order in which information ought to be entered. Our preference for novice users is to follow a top-down design philosophy where users first complete the statement part at any given Pascal scope, before dealing with the "details" of the declaration part. Variables generally are used, procedures and functions are called, before they are declared. But subprogram headings can be generated automatically when calls are first made. MacGNOME environments, however, do not restrict users to any fixed form of entry, allowing completely random positioning of the cursor with a point device. The user is also able to customize default cursor movement (and *many* other aspects of the user interface) to suit his own preferences.

Program Construction

The user must have some way of communicating the choices of construction to fill in the templates provided by the structure editor. Menus and/or other windows are an obvious place to enumerate valid choices at any particular stage in the program construction process. They also provide a mechanism for data entry that need not require active textual input by the user. In MacGNOME, for example, a mouse click on the appropriate option either expands the user's choice further, and/or enters the appropriate code. In addition to providing for the addition of new statements, expressions, and so on, MacGNOME's internal program representation allows a "list" menu which can be used to remind the programmer of user-defined names by scope, and to enter them where appropriate. We also allow the user to modify any logical entity in a program using a text editor.*

Error Reporting

The manner in which errors are reported often proves to be a great source of confusion for non-expert programmers. We have found that it is useful to report errors at several distinct phases in the programming process. An editor ought to prevent the user from making certain errors in the first instance. For example, if the user tries to declare the "same" identifier twice in the same scope, MacGNOME will refuse to accept the declaration. Some illegal assignments are also spotted at this stage. Other errors, and style warnings are reported immediately on exit from the scope of a procedure/function. Run-time errors must also be reported when they occur in an executing program.

Typically, error messages are cryptic at best. When an error occurs, the system must inform the programmer, but it ought to be done with a clear one or two sentence error message, such as "The variable 'SuchAndSuch' is used but never declared in PROCEDURE 'SomeName'." In our work with GNOME, we have catalogued the errors most commonly made by novice programmers. In MacGNOME, we are providing further suggestions about the most likely causes of the error in question. Associated with the most common errors, a few short paragraphs of text are available about what probably went wrong.

IMPLEMENTATION OF STRUCTURE EDITING ENVIRONMENTS

The wide variety of structure editing environments built over the past few years differ widely in the way they may look to the user, but internally they must all provide certain basic mechanisms to support an integrated environment for programming. In the next several sections, we outline some of the important design issues in implementing those mechanisms and sketch what we believe to be reasonable approaches to the problems they present. The primary consideration is to derive solutions which represent an acceptable compromise between functionality, generality, and ease of implementation.

The basic mechanisms can be generally classified by the need to support certain fundamental requirements of structure editing. These are:

1. *The need to represent trees and tools.* We must decide on an appropriate data representation for programs and provide some way to associate tool operations with various events in the system.
2. *The need to support checking and running of progams.* We must decide how frequently errors can be checked and reported. We must give some way to run and debug a program or some part of it.
3. *The need to display trees, program state, etc.* We must decide how the abstract representation of the program is to be mapped to concrete text, and how to make this run quickly.

We now consider each of these areas in turn.

THE KERNEL

At the center of every structure editor is a tree-oriented database. Because this tree database is the primary *static* data structure of the system, care must be taken in its design so that it may be useful to all the tools in the environment. Moreover, *dynamically*, it must facilitate the integration of many tools so that all the right routines are invoked at the right time.

* If the user opts to enter code in text mode, the system is released from the responsibility of presenting valid options. But, the error-reporting mechanisms may have to work overtime.

Tree Representation

Because the entire system operates on syntax trees rather than on text, it is crucial that the representation chosen for this database be easy for an implementor to understand and use when building the various tools that make up the entire environment. An implementation goal for us has been to keep the static data structures as simple as possible, and to compute dynamic properties of programs as needed rather than incrementally. Our emphasis on "light weight" trees is in contrast to other structure editors that incrementally maintain semantic information threaded through the tree. An example of this approach is our implementation of identifier definition site binding. (This will be discussed in a later section.)

Thus, the database elements, called *TreeNodes,* are simply linked tree nodes with a tag field that points to the appropriate grammar production description. While the basic structure of the database remains simple, our implementation also provides flexible interaction with a multitude of tools. This flexibility is supported by a LISP-like property list field in each TreeNode into which tools can store specialized information.

Integration and Object Oriented Philosophy

The static representation of the database must not only be clean, but the kernel is required to orchestrate the random and frequent incremental processing that is characteristic of interactive environments. To foster tool integration we have adopted an object-oriented pespective. By *object oriented,* we mean that we view each node in the tree as an *instance* of a *class*.[10] The class plays the role of a type in a standard high level language, determining what static fields are available in each node, and what functions are defined over that type. In addition, classes are arranged in a hierarchy so that behavior specified once can be shared by each class that is declared to be a *subclass*. Thus, a subclass refines the behavior associated with its parent class.

The most general class in our system is the TreeNode. Any behavior that the implementor needs to apply to every node in the system can be specified by including it in the class specification of TreeNode. At the next more specific level, we distinguish between *NonTerminals, Terminals* and *Metanodes.* Finally, refining NonTerminals, we get to classes that represent individual productions of the language. Thus, a particular IF node is an instance of the class **IF.** Via inheritance it behaves like any other **NonTerminal,** as well as a **TreeNode.** (See Figure 2 for a graphic description of this hierarchy.)

While the class hierarchy groups together node types that behave similarly, the actual internal events that occur on nodes[†] are defined around the tool-independent events CREATE, DELETE, EXPUNGE, and MODIFY. Behavior can be associated with the tuple **[EVENT, CLASS].** For example, when the identifier of a type declaration **(TypeId)** is deleted, the code associated with **[DELETE, TypeId]** is executed. In addition, code associated with its superclass **(IdDef)** is invoked through the tuple **[DELETE, IdDef].**

The Benefits of This Approach

The object-oriented programming style seems to be a particularly effective integration technique for interactive programming environments. An interactive system is characterized by multiple subsystems, each contributing conflicting requirements, and unpredictable sequences of state changes. *Ad hoc* integration techniques usually fail to produce flexible and robust interactive systems.

We have found the class hierarchy to be an effective way to organize these requirements, allowing sharing where appropriate. Each tool or subsystem can group together classes of nodes that behave similarly, allowing for a compact description of the behavior of any node.

The availability of the class hierarchy and the identification of the basic internal transition events has led to a disciplined approach to new tool integration. To add a new tool, the implementor must decide how his tool is to act in response to each of the basic transition events. The system guarantees that if his tool can handle any transition, it will work in the face of unpredictable events triggered by user editing commands and the integration of other tools. The class hierarchy gives the implementor the ability to share specification of behavior among nodes, thereby simplifying his problem of specifying the behavior of *every* node.

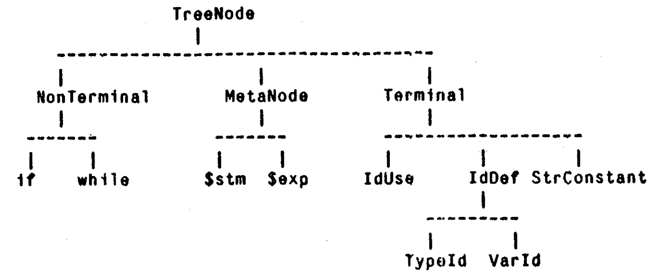


Figure 2—The class hierarchy

[†] In object oriented terminology, the "messages sent to the instances."

An Implementation: Getting Pascal To Do It

Fortunately, we can reap the benefits of object-oriented programming without the cost of implementing a full-blown object-based system. This is done by exploiting our stylized and well-defined use of objects. Rather than treating every data structure as an object, only TreeNodes are treated as objects. Because the class hierarchy is known in advance, we can coalesce the memory requirement for a TreeNode into a simple Pascal variant record (see Figure 3). Finally, since all message names are known, we can simulate "method lookup" by using case statements.

For example, in Figure 4, sending the message **DELETE** to an arbitrary object is handled by a routine **XXDelete** that glues together all the code pieces specified for each tuple matching **[CLASS, DELETE]**. In this case, deletion of a type definition causes its name to be removed from the menu of available types in addition to any other action specified for TreeNodes in general.

The real benefit of the object-oriented approach has been the order it imposes on the complexity of an interactive system consisting of a large variety of node types. It has enabled us as implementors to formalize our task: *for each subsystem, and each node type, completely specify its behavior for each internal transition event,* and to simplify its solution *by grouping common behavior into a single piece of code, and knowing that the Kernel guarantees to invoke the code appropriately.*

DISPLAY

While the internal representation of a program is a tree, the tree must be displayed as text and graphics on a screen in order for a programmer to view it. The process of mapping the abstract internal structure to concrete, visible forms is often called *unparsing,* and the part of the kernel devoted to this operation is called the *unparser.* The quality of the interaction between a user and the programming environment will depend largely on the ability of the unparser to present flexible, powerful, and insightful projections of the underlying structure. As such, the unparser is one of the critical components of design.

The User's View

There are four important display issues for the user:

1. *Appearance.* Programs should make appropriate use of fonts and font styles, indentation, and graphics. Formatting should be sensitive to the amount of space available; it should be possible to elide* parts of the program, and the representation of nodes should conform to window widths.
2. *Response.* Small changes must result in quick updates. While a user may be willing to wait for the system to respond to major global changes, small local changes need to be reflected immediately.
3. *Functionality.* The display must support the usual window operations: scrolling, paging, mouse selection, window dividing, etc. It is also extremely valuable for the user to be able to edit textual items "in-place." For

```
TYPE TreeNode = RECORD
      classtag        : ClassKinds;
      father, sibling : ↑TreeNode;
      propertylist    : ↑AttributeValuePair;
      CASE Shape : ImplementationKind OF
            NonTerminal : ( children : ↑TreeNode );
            Terminal    : ( value    : string );
   END; { TreeNode }
```

Figure 3—The TreeNode record type

```
PROCEDURE XXDelete( aNode : pTreeNode; aTag : ClassKinds );
BEGIN
    {First, handle the most specific CLASS }
    CASE aTag OF:
        TypeId: {Code for [DELETE,TypeId]}
            RemoveNameFromTypeMenu(NameOf(aNode));
        IdDef:
            {Code for all Identifier Definitions on DELETE.}
        TreeNode:
            {Behavior common to all TreeNodes.}
    END;{CASE}
    { Now, let the SuperClass also be notified }
    IF IsDefined(SuperClassOf(aTag)) THEN
        XXDelete(aNode,SuperClassOf[aTag]);
END;
```

Figure 4—The DELETE event dispatcher

* By *elide* we mean the ability to suppress visual details. The elided form of "IF a < b THEN x: = 2;" might be "IF a < b THEN. . . ."

example, to change the phrasing of a comment, the user should be able to point to the comment and simply edit it as he would edit any piece of text in a text editor. Finally, it should be possible to tailor the details of appearance to personal taste. For example, one user might prefer Pascal keywords to be italicized while another might prefer bold face.

4. *Views.* It must be possible to view the program from different perspectives. For example, a *documentation view* might suppress details of the code; a *code view* might suppress details of documentation.

The Implementor View

For the implementor, the requirements of flexibility and efficiency itemized above present a significant challenge. In particular, there are two design issues to resolve: the kind of language that the implementor will use to express the desired formatting and conditionality, and a way to make it run fast. We now sketch the approach we have taken to these two issues. (For a more thorough treatment see Garlan.[9, 11])

The unparse specification language

The language for specifying the representation of a node must allow the implementor to attach to each node a *collection* of representations that represent the way the node will be formatted in its various contexts. Multiple views and conditional formatting imply the need for multiple representations. This in turn requires some way to specify how the format for a node is to be chosen from the many alternatives. The problem of control of unparsing presents two conflicting goals, *first* because there is a hierarchical relationship between nodes, a node must be able to influence the choice of representation of nodes in its subtree. For example, if the user indicates that a procedure is to be elided, all the nodes in the procedure subtree must be informed of the fact, and *second* because the display of nodes is quite idiosyncratic, the node itself must be responsible for determining how it should look. For example, display of an IF-THEN-ELSE is quite different from the display of a WHILE-DO. Somehow, an unparse scheme language must allow both kinds of control to take place.

Our solution is illustrated in Figure 5, which shows the unparse specification for the IF-THEN node is in the *code view.* The important points to note are as follows:

1. Multiple views allow different such specifications. Thus a *documentation view* of this node would have a completely different set of formatters.
2. For a given view, the representation of the node is determined by a set of condition-action pairs. When the node is to be unparsed, each *condition part* is evaluated. The first condition to succeed triggers the corresponding action, or *scheme part* of the specification. (Since the last condition is always "TRUE," one scheme will always succeed.)
3. The scheme part of the specification itself allows space-dependent conditionality. In the example, the "Horiz-Vert" notation indicates that the IF will be formatted horizontally if there is room; vertically otherwise.
4. The condition part of the scheme can make use of *perspectives* such as *ellipsis.* These perspectives are passed down from the parent node, thus allowing one node to influence the behavior of its subtree while maintaining autonomy at the node itself. In the example, the ellipsis perspective causes the IF node to be elided.
5. The values of attributes of a node can influence how it is displayed. In the example, the *err* attribute of the node causes a special kind of highlighting (defined by the implementor). Because comments in our system are treated as attributes of a node, the existence of a comment on the IF-THEN node causes the comment to be displayed in the documentation view of the program.

The behavior of a node embodied in this style of unparsing is very much in keeping with the object-oriented approach discussed above. Nodes are viewed as individual and idiosyncratic objects which have considerable freedom in determining their individual behavior.

```
Node IF-THEN  ::= cond: bool-exp then-part: statement

    Default View code

        Schema:       ellipsis ==> "If <cond> Then ..."
                      .err     ==> "@myhighlight(If <cond> Then <then-part>)"
                      TRUE     ==> "@horiz-vert((If <cond>)(Then <then-part>)"

    View documentation

        Schema:       .comment ==> "<.comment>"
                      TRUE     ==> ""
```

Figure 5—Unparse specifications for the IF-THEN node

Efficient implementation

Implementations of unparsing can be characterized by the degree of incrementality that they allow. Early unparsers redisplayed the entire tree after *every* change to the tree. Typically, they would start at the root, interpreting the scheme associated with that node, and recursively each subcomponent of the node. While this has the advantage of being easy to understand and implement, it is unforgivably slow on large trees.

A second generation of unparsers sought to improve performance by unparsing only a portion of the tree. Ideally, only the nodes that would actually be visible would be reunparsed after each change. This gives acceptable performance to unparsing, but requires that the unparser go through some curious maneuvers to figure out where to start unparsing. The problem with this approach is that unparse schemes had to be relatively simple in order for this calculation to be reasonable. In particular, the kind of conditionality that we outlined above is quite beyond the abilities of this strategy.

A third technique, the one we have adopted, only updates the screen for nodes that have actually changed. In order to make this extreme form of incrementality possible, we cache the information from previous unparsing in an *unparse tree* (U-tree). The U-tree encodes all the concrete syntactic information associated with the current view(s) of the abstract tree. When changes occur in the abstract tree, the corresponding subtree of the U-tree is recalculated. Further dependencies within the U-tree are then allowed to propagate. Finally, some portion of the U-tree is written to a buffer for display to the screen. The calculation of the amount to be written to the buffer is now easy because all the size information has been computed in the U-tree.

SEMANTICS

While the syntactic and lexical guidance provided by a simple structure editor is extremely helpful, it still does not aid in detection and correction of *semantic* errors. There are three classes of semantics that require attention:

1. *Static semantics*—semantics that can be done at compile-time; for example, type checking of expressions, and identifier resolution.
2. *Runtime semantics*—semantics that depend upon the running state of the program tree; for example, checking that array index references are within bounds.
3. *Environmental semantics*—semantics that characterize the relationship between the interaction between the program editor and the editor environment; for example, user-settable key bindings, positioning of windows, etc.

Static Semantics

The choice of mechanisms to support static semantic checking is closely tied to the policy issue: *What grain of incremental checking should be provided?* There is a wide spectrum of choices. At one extreme, the editor could provide semantic feedback *batch* style as in a traditional compiler environment. Here the entire tree is reprocessed when checking takes place. Consequently, checking is usually done for large grain sizes. At the other extreme, *every* change to the program tree could cause incremental rechecking of the changed pieces. For the *implementor*, the former choice may appear attractive because it allows him to integrate an existing stand-alone compiler directly into the editing environment. By contrast, the latter choice requires close cooperation between the editor and the semantic analyzer, and so a special-purpose semantic tool must be integrated into the system. However, for the *user,* the batch method can be awkward and even painful; there are frequently long delays between any change and rechecking of the program, and lists of errors can become quite long and may have little relation to the logical structure of the program since they are frequently organized by line number. On the other hand, incremental error reporting at the finest grain can be intrusive and misleading. For example, if the user programs by first using procedures and then defining them, it can be quite annoying if the system keeps reminding him of the undeclared items.

An appropriate solution is one which provides a flexible enough underlying mechanism such that policy can be tuned for maximum benefit and minimum frustration. In doing this, the mechanism may not support either extreme, but will provide an efficient implementation for a wide range of grain sizes of incremental semantic interaction.

In implementing incremental semantic processing, there is an important tradeoff between the amount of incremental semantic information, *state,* that is stored and the amount of recalculation that is done when the program is to be checked. In the previous two examples, the batch method requires no saved state and does all checking from scratch each time while the highly interactive method benefits from as much semantic state as can be made available. Unfortunately, the more state that is stored, the more computational overhead is required to maintain it during editing. For example, if the definition site of an identifier is kept with every one of its use sites, any change to the definitions in an enclosing scope requires updating all use sites.

We have found that by making a judicious choice of the information stored after each semantic check, it is possible to provide highly incremental semantics with a minimal amount of incremental overhead. We now illustrate how this is done with respect to identifier resolution for a Pascal environment.

In order to perform most of the static semantic checking for Pascal, it is first necessary to associate uses of an identifier with their definition site. This might seem relatively easy to do; after all, Pascal was designed to be compiled in one pass. But in an interactive environment, we cannot afford to process the complete tree (batch style) every time any semantic checking takes place. One solution to this problem is to calculate the definition site incrementally with every use and store the value with every use site. But this has the disadvantage, mentioned above, that any change to variable declarations requires a complete reprocessing of the affected scope.

An alternative solution that we have adopted keeps *no* use-definition associations with definition sites, but provides a simple mechanism for a quick mapping from an identifier use to its definition site. The mechanism is based on the fact that at any point *P* in the tree, the naming environment at *P* is completely determined by the set of modifier nodes on the path between *P* and the root of the tree. The modifier nodes include the "." (field selection) operator, the WITH statement, and any declaration lists. The naming environment of *P* can thus be taken to be a *stack* of modifiers. Consequently, to resolve a use site of an identifier, we initialize a stack of modifiers by walking up the chain of fathers of the use site, pushing an appropriate modifier representation as we encounter it. Once this is done, the use site can be resolved by scanning through the stack until a modifier is found that defines the current use. Subsequent identifier resolutions need not reinitialize the stack each time, but the modifier stack is kept current as the semantic analyzer moves from one node to another.

The success of this technique depends on the fact that Pascal program trees are not very deeply nested, that only a few nodes in the tree can modify the naming environment, and that the information stored at each modifier is in an efficiently usable form. The last criterion is met by allowing each modifier to be, in effect, a symbol table. For example, the modifier for a WITH statement would be a symbol table containing the fields of the record type specified in the WITH statement. In this way, pushing a modifier on the stack involves merely copying a pointer, and resolving an identifier requires looking up the name in each modifier symbol table until one is found that defines it.

In a sense, we have a semi-batch solution to the problem. Because name look-up is cheap through the calculation just outlined, we can avoid the overhead of fully incremental semantic state and still achieve a wide range of policies. Experience has shown that the efficiency of this technique is more than adequate for all but the smallest grain sizes of incremental checking, and for all but large, heavily nested programs. The important general lesson to be learned from this is that it is not worthwhile to incrementally maintain information that is easy to regenerate on demand.

Runtime Semantics

For the user, the following are legitimate expectations of a runtime mechanism in an integrated system:

1. It should provide reasonable execution speed for debugged code.
2. It should support source level debugging.
3. It should give the user the ability to edit source code *during* the execution of a program.

The existence of the syntax tree as an underlying representation of a program makes it possible for an implementor to provide these facilities in an integrated system. We illustrate how these goals can be met by mentioning three details of implementation.

1. We allow a mix of compiled and interpreted code. To make this possible, compilation takes place at the grain size of procedure. Procedure calls are implemented by an indirect call through a jump table. Each entry in the table indicates whether the procedure is to be interpreted or executed as compiled code. Compiled procedures can then be replaced by their "interpreted" counterparts, or by new compiled versions simply by modifying the procedure jump table.
2. Interactive editing during execution takes place on the syntax tree itself. For example, breakpoints are set by adding a special BREAK statement to the source. (See Feiler and Medina-Mora[12] for more details on how this is done.) Modified procedures can then be interpreted or recompiled as desired.
3. To limit the amount of rechecking needed after a user makes a change to the source, we maintain with each procedure or function a list of non-local identifiers used. Thus, when a variable's declaration in a procedure is modified, we need only recheck nested procedures that actually use the affected variable, not *every* nested procedure.

The combination of these features can provide a powerful working environment for the user. A typical scenario might proceed as follows: The user starts out by building a simple program, one procedure at a time. After each procedure is built, it is tested by running the program with a testing call to the new procedure. At this point, the user would "run" his code using the interpreter in order to provide a quick run-edit-run cycle. Upon verifying these first few procedures, the user might tell the system to compile them, and move on to other tasks. Then, the program would run in a mixed mode, executing directly-compiled procedures while interpreting the newer code. This process would continue until the entire program is debugged and ready, and would result in a totally compiled program. However, a procedure may be edited and interpreted at any point in order to use the breakpoint facility or the source level debugger (the debugger would also work on the compiled code, but with a limited functionality).

Environmental Semantics

The interaction between the user and the program can be seen as another set of semantic interactions within the program. This notion is quite powerful in that it abstracts the real-time interaction away from the core functionality of the program. This enables the internal mechanisms to function solely based on state and goal information.

For the implementor, the operative words are *extensibility, predictability,* and *flexibility.* We have found that the most useful implementation is one whereby all user interactions flow through a *dispatcher.* This clearing house for commands keeps the external interactions well removed from the internal operation, and allows the user interface to be incrementally adjusted as development of the environment progresses. In our system, all internal operations are defined independently of the way they are invoked in the user interface; for example, it could be from the keyboard, a file, the mouse, or any input device. The *user* is allowed to define the mapping from real-world events to internal events. Of course, it is necessary to establish defaults of various kinds, particularly in an environment meant for novice users. The novice need not even be aware of the features that can be changed, but we believe it is important to allow considerable flexibility for those who wish it. It is possible to anticipate and accommodate all sorts of preferences ranging from fonts and highlighting to cursor movement. Menus and interactive windowing, among other methods, make such changes easy, and can be saved in a user's "profile" for future use.

CONCLUSIONS

The use of integrated programming environments based on structure editing is an emerging technology that has reached the stage of being both demonstrably useful and readily implementable. We have outlined some of the salient aspects of our work in developing such environments, and suggested paths of implementation that seem to be worth traveling. A predominant theme has been the need to separate policy from mechanism. While the choice of user interface policies will probably differ widely from those presented here, the mechanisms that we have sketched nonetheless will be applicable to future environments.

One of our primary motivations in developing GNOME and MacGNOME has been the desire to improve the learning environment for novice programmers. GNOME has already had a marked impact on our students. They are learning more, in less time, than they did a few years ago. Moreover, we think that they are approaching programming in a different manner following the deemphasis on details and emphasis on modular problem solving that the programming environment encourages.

Syntactic errors are not possible. Almost all semantic errors are either prevented from being made, or are caught in an incremental manner, immediately or on exit from scope when constructing the program. Students thus are unlikely to accumulate huge numbers of seemingly unrelated errors. Errors are reported in context by scope, with the offending code highlighted in inverse video. Messages are less cryptic than one sometimes sees. Particularly useful editor commands in GNOME include those that allow for easy checking for semantic errors by scope at any time, and for consistency between actual and formal parameter specifications.

Because we have been developing these environments incrementally at Carnegie-Mellon, we have not yet made a formal assessment of their impact. We are planning to test the GNOME environment elsewhere, where we can take advantage of better quasi- and field-experimental control conditions. Much of our evidence about GNOME's effectiveness is based on our informal impressions. Admittedly, other factors have also changed over the period during which we have been developing these environments. Still, that period has been concurrent with a marked rise in student performance on their mastery examinations. That is compelling evidence to us, particularly since it has occurred in spite of the fact that our student body has increased in size and heterogeneity.

ACKNOWLEDGMENTS

Thanks are due particularly to Lee Miller and Robert White for their many contributions to the ideas developed in this paper.

REFERENCES

1. Habermann, A. N., and D. Notkin. "The Gandalf Software Development Environment." Technical Report, Carnegie-Mellon University Computer Science Department, January 1982.
2. Medina-Mora, R. "Syntax-directed Editing: Towards Integrated Programming Environments." Ph.D. thesis, Carnegie-Mellon University, March 1982.
3. Donzeau-Gouge, V., et al. "Programming Environments Based on Structure Editors: The MENTOR Experience." *Technical Report 26,* Institute National de Recherche en Informatique et en Automatique, France, May 1980.
4. Teitelbaum, T., and T. Reps. "The Cornell Program Synthesizer: A Syntax-directed Programming Environment." *CACM,* 24(9), 1981.
5. Reiss, S. P. "Graphical Program Development with PECAN Program Development Systems." *Proceedings of the Software Engineering Symposium on Practical Software Development Environments.* ACM-SIGSOFT/SIGPLAN, April 1984.
6. Garlan, D. B., and P. L. Miller. "GNOME: An Introductory Programming Environment Based on a Family of Structure Editors." *Proceedings of the Software Engineering Symposium on Practical Software Development Environments.* ACM-SIGSOFT/SIGPLAN, April 1984.
7. Jensen, K., and N. Wirth. *Pascal User Manual and Report (Second Edition).* NY: Springer-Verlag, 1974.
8. Pattis, R. E. *Karel The Robot: A Gentle Introduction to the Art of Programming.* NY: John Wiley & Sons, 1981.
9. Garlan, D. B. "Views for Tools in Software Development Environments." Ph.D. thesis, in progress, Carnegie-Mellon University.
10. Goldberk, A. J., and D. Robson. *Smalltalk-80: The Language and Its Implementation.* Reading, Mass.: Addison-Wesley, 1983.
11. Garlan, D. B. "The VIZ Unparse Specification Language and the VAL Unparser." Technical Report, Carnegie-Mellon University, June 1985.
12. Feiler, P. H. and R. Medina-Mora. "An Incremental Programming Environment." *IEEE Transactions on Software Engineering,* SE-7(5), September 1981.

Effects of microcomputer technology on young handicapped children

by PATRICIA L. HUTINGER and MARILYN D. WARD
Western Illinois University
Macomb, Illinois

ABSTRACT

Project ACTT (Activating Children Through Technology) is a handicapped children's early education program (HCEEP) model demonstration, housed in the College of Education at Western Illinois University. Its major goal is to develop, implement, and demonstrate an innovative microcomputer curriculum model using affordable and practical microcomputer hardware and software. The primary target population is composed of children from birth to six years of age who demonstrate moderate to severe structural and functional handicapping conditions that prevent them from interacting with their environment. The families of these children are also major targets for intervention and are partners in Project ACTT.

INTRODUCTION

Although microcomputer technology is not new, its application for handicapped children is relatively recent. It is known that handicapped children can benefit from educational use of microcomputers just as nonhandicapped children can—perhaps in even more dramatic ways. Curriculum materials for the microcomputer, paired with the potential of the intelligent machine itself, promise to facilitate the learning of the handicapped well beyond any other advance in instruction to date. However, application of the potential of the microcomputer to the education of young handicapped children is in its infancy. The experimental work that exists, such as that done by Brinker and Lewis,[1] is not yet practical in terms of cost. The development of practical, affordable applications is still to be completed. Appropriate software is virtually nonexistent, and alternative input devices, although available, must be adapted for use in programs for handicapped children.

Despite these difficulties, the potential of the microcomputer and its accompanying software holds great promise for young children, particularly severely handicapped children. Microcomputers are endlessly patient; they can be programmed to respond in many different ways to diverse forms of input and to function in an interactive fashion. The microcomputer may very well transform education programs for young handicapped children, just as it has affected the rest of society. In order for this transformation to occur, the appropriate software must be developed, hardware and firmware must be selected and adapted, and professionals must know where to begin.

Using Apple II microcomputers, Brinker and Lewis developed a contingency intervention system (based on Piagetian theory) to foster within handicapped infants the expectation that the world is controllable, and to lead them to user-specific behavioral movements to explore the contingencies available. Contingency intervention is defined as the arrangement of events that can be consistently controlled by an infant's behavior.[1] The rationale is that handicapped infants often are at risk of being deprived of contingency experiences because exploration of contingencies depends on motor responses and available social contingencies, which are in many cases minimal, owing to the family's low expectations for the child.

Children who do not have contingency experiences early in their lives may develop the attitude that their situations and the environments within which they find themselves are in no way under their control; that is, there is an external locus of control or they experience learned helplessness. This sort of psychological characteristic is extremely difficult to modify, particularly when the conditions that cause the attitude persist for a long period of time, as is typically the case for high-risk and handicapped infants. The results of this externality of learned hopelessness include retarded learning rates, lack of self-direction, limited goal-oriented behavior, and disturbed social interaction. All of these difficulties, of course, compound the problems of handicapped individuals.

Microcomputers have been used effectively to individualize instruction for elementary age and older children in a wide variety of teaching modes, including drill and practice, educational games, direct teaching, simulations, and problem solving. Many of these applications also can be used with younger handicapped children. The effectiveness of microcomputers lies in their multisensory potential; their ability to store, retrieve, and use data on children's programs; and their ability to actively involve children in the learning situation.[2]

The use of technology in special education is, of course, not new. Manipulative devices and sensory equipment have been used for years with handicapped children. Hannaford and Sloane point out that, "As a profession, we have been quick to see the potential of these technologies for the learner with special needs."[2] However, tape recorders, projectors, teaching machines, and switches are a far cry from the complex machine intelligence available with microcomputers. Additionally, microcomputers are the first technology that can be used by teachers of young handicapped children that also play an essential role in adult society in exactly the same form.

Although educational and intervention programs to serve young handicapped children, beginning as early as birth, have developed at an increasing rate over the past decade, until now no program has made full use of the technology offered by the microcomputer revolution. Inexpensive, powerful machines are now widely available in homes, businesses, agencies, and schools. However, most microcomputer applications in special education can be classified as used in management (for keeping records and charting students' development), assessment, strategies for physically handicapped individuals, and drill and practice for school-age children (Bennett, 1982).

PROJECT ACTT

Project ACTT (Activating Children Through Technology) is a handicapped children's early education program (HCEEP) model demonstration project funded by the U.S. Office of Education and housed in the College of Education at Western Illinois University. The project's main goal is to develop, implement, and demonstrate an innovative microcomputer curriculum model using affordable and practical microcomputer hardware and software. The primary target population comprises children from birth to six years of age who demonstrate moderate to severe structural and functional handicapping conditions that prevent them from interacting with their environment. The families of these children are also major targets for intervention and are partners in Project ACTT.

ACTT is a rural project, developed and demonstrated in west-central Illinois. It incorporates the direct efforts of five agencies: Western Illinois University, McDonough County Rehabilitation Center, Bushnell–Prairie City Community Unit District #170, Colchester Community Unit District #180, and Hancock Central Unit District #338. The microcomputer curriculum intervention developed by the project is being used with the target population of children served in the McDonough County Rehabilitation Center 0–3 program and the 3–6 public school programs in the two school districts. Home visits are part of the service provided by the 0–3 project; however, children in this age bracket also are brought to the ACTT center for microcomputer intervention. The teachers in programs for three- to six-year-olds have been trained to use the microcomputers, and ACTT staff work in their classrooms at least once a week.

The ACTT project staff is using microcomputers with handicapped children as young as eighteen months and plans to work with even younger children. The present caseload includes children who function developmentally under 12 months of age. The major emphasis is on helping children develop a sense of autonomy and control over their environment—something that young handicapped children seldom experience. Families participate in the ACTT activities at different levels of involvement. New software is being developed and existing software is being customized for use with young handicapped children. Adapting selected hardware and peripherals to the needs of these children is also part of the ACTT project.

The ACTT Curriculum

The ACTT curriculum has been developed according to three priorities. First is the need to foster the child's expectations that he can control his environment. Various recordkeeping software was also developed at this stage. Second, the curriculum had to provide a mode of communication. Finally, the curriculum was to develop selected preschool skills for children from birth to six years of age. At present, software in the first and second categories has been developed. Planning for the third is now in process; curriculum activities using commercial software are already available.

The ACTT curriculum is based on the knowledge that young handicapped children are limited in the amount of interaction they can have with their environment. The initial interaction, or lack of it, with the environment is crucial in the development of the child's sense of autonomy and problem-solving strategies. Microcomputer technology provides the means for the young child to control aspects of the environment using switches, adaptive peripherals, and software that have been customized to meet individual needs.

Curricular objectives for the 0–3 population have been designed for use with the Macomb 0–3 Rural Project's Core Curriculum, a functional curriculum widely available throughout the country.[3] Goals and objectives reflect a functional approach and include fine motor, gross motor, social, cognition, language (including an emphasis on pragmatics), and self-help skills. The curriculum was developed by a nationally recognized infant model project that has served hundreds of handicapped infants and their families since 1975. Microcomputer curricular activities are used with various adaptations as deemed appropriate.

Preschool curricular objectives have been developed by the ACTT staff.[4] A progression of developmental behavior associated with acquisition of LOGO skills is currently being tested.[5] Other objectives, related to a wide variety of preschool goals as suggested by Neisworth and colleagues in the HI-COMP curriculum,[6] are related to cognition, problem solving, gross motor skills, fine motor skills, language development, social skills, and self-help skills. The ACTT curriculum is designed to be cross referenced with the HI-COMP curriculum and includes objectives and activities related to attention; concept formation; creativity; interacting with new activities; and experimenting with color, line, form, and space using computer graphics. Customized applications include the use of switches, keyboards, voice synthesizers, robots, and other peripherals. Specific adaptations that must be made with commercial software are part of the project's scope. Review of appropriate commercially available software is under way.

Switch-activated toys have been used to foster cause-and-effect concepts in children from birth to three years of age. These toys are connected to the computer in an effort to monitor responses, add additional capabilities to the switches, change reinforcers, and help the children become accustomed to using a switch—a skill that leads to more sophisticated computer use. Several software programs have been developed to teach children to use switches to control visual features or sound in a program. In addition, some commercial software has been customized by adding preboot disks for single-switch input. A program used in conjunction with a voice synthesizer, which "says" the name of an object when the child touches the appropriate picture on a graphics tablet, also has been developed.

A variety of commercially available software, including LOGO, and project-programmed software is used with the 3–6 population to reinforce the concepts of environmental control and problem solving. Special switches are used by children who cannot manipulate the keyboard. Children operate a robot with a customized version of Instant LOGO and a modified graphics tablet.

The 3–6 population has used LOGO extensively because it is designed to help children acquire skills that are important to ACTT goals. Adaptations to allow for alternate input have been developed, but all children do not need to use them. A number of precomputer activities also are used prior to starting children on the computer. A detailed LOGO curriculum is now being developed.[7]

The effects of the ACTT curriculum on the children are being examined carefully. Videotapes of sessions with severely handicapped and 0–3 populations are taken for later analysis. The project is implementing single-subject design studies to determine the various factors that hinder or help each child's progress. Questions related to child preferences for varying software and firmware characteristics, as well as optimal time and content of computer sessions, are being studied.

In addition, an observational instrument, the BIT (behavior interaction tool) was developed for use with individual children or groups of children and is currently being field tested. A program to record and analyze behavior has been coded for the Epson HX20. Data files are sent by modem for storage on a mainframe. Results are expected to provide important information about what young handicapped children actually do when they use a computer. The BIT allows collection of data regarding individual child behavior, child–peer behavior, child–adult behavior, and adult behavior.

Physical Equipment

The Apple II+and Apple IIe were selected for use in the ACTT environment. The decision to use them was based on the variety of available peripherals and software. Resources dictated the choice of only one type of computer because funds were not available to purchase different computers and their accompanying software. Because some applications involve internal machine modifications it was important to purchase equipment that could be interfaced easily with various peripherals for specific purposes. A series of input–output boxes has been constructed to make use of the potential of the gameport for alternate inputs. These boxes can be used to accept single or multiple switch input in place of game paddle or single-key control.

One of the microcomputers used with the I/O box also includes an Adaptive Firmware card, an Echo II or a Votrax, a Unicorn keyboard, a graphics tablet, and a Gibson light pen. A RAM card with 64K memory has been added. This computer does not hold all peripherals at one time, but all are available if needed.

In preschool classrooms the computer ordinarily is set up in a corner, much as any center-of-interest activity would be. Its use becomes one of a number of choices children make during the day. Children often work in groups of two or three, helping each other decide what direction to make the turtle go as they work with LOGO. An effort to use the computers in ways that encourage social interaction is a major consideration in any ACTT activity.

Teaching Strategies

Although some ACTT activities involve direct instruction, many involve setting up a situation in a way that will lead the child to control his own learning. LOGO activities necessitate changing teaching style. Many more divergent questions directed toward problem solving and descriptive comments are asked when children use LOGO. A great deal of planning goes into the development of instructional activities.

The developmental level of computer use seems to involve precomputer activities, transition into computer activities, computer activities, and follow-up activities related to curriculum integration. Teaching strategies may differ from one level to the next.

An essential instructional condition involves the child's control of the computer. This means the teacher does not spend all the time demonstrating while the child watches. In fact, the reverse is true. Frequently, one child operates the computer with advice from other children. Preschool children are often grouped according to various factors staff believe will help them to work well together in this manner.

Case Studies

David, a five-year-old with cerebral palsy, is confined to a wheelchair. He has head control, good speech, and a cheerful sense of humor, but he does not have functional hand control, nor does he pick things up and hold them. His vision is restricted in ways his parents, teachers, and other professionals have not yet determined, but David can control a Topo robot with Instant LOGO and a Koala Pad to move Topo around the room in ways he plans ahead of time. He knows that he is in control; he can stop and start the robot at will. With an adaptive keyboard, David can write stories with Storymachine, and he can play Dragon's Keep if someone reads the text to him. Using special software, he can touch the keyboard to make sounds; either music or synthesized speech. He can work through the First Words program by himself. He is learning, with his mother and often with his father, to work through a variety of computer activities to help him learn to use the computer as a tool and to function within the kindergarten setting.

Cheryl, a six-year-old who has not yet been diagnosed accurately, has frequent seizures, is often unaware of the surrounding environment, and has poor language skills. However, she uses precomputer activities in a way that may lead to greater communication with the outside world. Although they started with switch-controlled toys, Cheryl and her mother now use an adaptive keyboard and produce music or sounds. Cheryl pays attention to these activities for increasing amounts of time, smiles frequently, and demonstrates more receptive language than she was thought to have. Additionally, her sense of humor shows in her play with the switch-operated toys. She appears to understand that her hand pressure on a switch results in the movement of a toy rabbit or the barking of a toy dog.

Jerry, an eighteen-month-old who has cerebral palsy, cannot sit unsupported, but he can press a switch to change the Stickybear images on the screen. Anne is two and one-half years old and has little language, but she appears to have near normal physical development and is able to touch a picture on an adaptive keyboard and hear a voice speak the name of the object. Her mother is learning to use a light pen so she can help Anne draw on the screen. Anne's sense of affecting her own environment is not often apparent, but the computer's immediate feedback may help her feel that she does, indeed, make things happen.

Four-year-old twins John and Mark both have cerebral palsy. John is visually impaired and Mark is blind. Both twins function physically at a developmental level below one year, but they are socially aware and respond happily to other people. They have learned to use swtiches to make things happen on the computer—sounds, music, toy movement.

John presses a switch and watches Stickybear. Mark works with two switches, listening to two different kinds of sounds. The twins' mother works with an ACTT staff member weekly and their father comes to sessions when he can. Both parents believe that the computer is already helpful to their children and that it will be even more useful as they grow older. Meanwhile, the twins are confident they can do something to affect their own environment.

CONCLUSIONS

Young handicapped children can control their environment through computer use. Early intervention using the computer as a tool to help the child gain autonomy, develop problem solving skills, and acquire a sense of cause-and-effect is part of the curriculum being developed by Project ACTT. Materials presently in the production stage will be widely disseminated to help others accomplish similar work. Data collected will provide important information about the effects of technology on young handicapped children and their families. Careful application of the findings may be beneficial to a whole generation of children in the information age.

ACKNOWLEDGMENT

This work was developed with funds provided by a grant from the Office of Special Education Programs, Department of Education. The content of this paper does not reflect the position or policy of the U.S. Department of Education, and no official endorsement of this paper should be inferred.

REFERENCES

1. Brinker, R., and M. Lewis. "Making the World Work with Microcomputers: A Learning Prosthesis for Handicapped Infants." *Exceptional Children,* 49 (1982), pp. 168–170.
2. Hannaford, A., and E. Sloane. "Microcomputers: Powerful Learning Tools with Proper Programming." *Teaching Exceptional Children,* 14 (1981), pp. 54–57.
3. Hutinger, P., S. Marshal, and K. McCartan. Macomb 0–3 Core Curriculum. Macomb, Ill.: Outreach Macomb 0–3 Project, 1983.
4. Harshbarger, K., L. Perry, L. Robinson, and A. Sutton. ACTT Curriculum. Macomb, Ill.: Project ACTT, 1985.
5. Harshbarger, K. Developmental Stages of LOGO. Macomb, Ill.: Project ACTT, 1985.
6. Neisworth, J., S. Willoughby-Herb, S. Bagnato, C. Cartwright, and K. Laub. *Individualized Education for Preschool Children.* Germantown, Md.: Aspen, 1980.
7. Perry, L. Getting Started with LOGO. Macomb, Ill.: Project ACTT, 1985.

Panel: Hardware/software standards in education—a manufacturer's perspective

Chair:
DAVID SPRAGUE, *Western Illinois University,* Macomb, Illinois
Members:
JOSEPH CASHEN, *Acorn Computer Corporation,* Woburn, Massachusetts
H. JAMES WATTS, *Apple Computer, Inc.,* Cupertino, California

Personal computers are becoming commonplace in our nation's schools. Most schools purchase systems based on educational software support; yet ironically, after the purchase, the two principal uses of the computers are for computer literacy and computer programming. What does this imply for the substantial third-party software base for personal computer manufacturers? Previously, manufacturers targeted schools as an indirect channel for reaching the home computer buyer. Currently, a hardware manufacturer's school support programs are as vital as the personal computers available in making the school sale. What does this mean for hardware/software standards?

What are the standards for educational hardware/software for the future? Who will determine them? Representatives of leading personal computer hardware manufacturers discuss the next generation of personal computers used in education.

Panel: Designing computer-based learning systems

Chair:
MARILYN D. WARD, *Western Illinois University,* Macomb, Illinois
Members:
STEPHEN FRANKLIN, *The University of California—Irvine,* Irvine, California
KATHERINE HARSHBARGER, *Western Illinois University,* Macomb, Illinois

The Educational Technology Center at the University of California—Irvine is one of the leading projects in the world for research and development of computer-based learning material. Stephen Franklin discusses strategies for designing curriculum materials to use with computer-based learning systems. This material is the result of work by Alfred Bork, Stephen Franklin, and others at the Center. Harshbarger and Ward provide commentary on those strategies, based on experience in using software with students.

Panel: Corporate policies and government legislation—support for education

Chair:
HAL BERGHEL, *University of Nebraska,* Lincoln, Nebraska
Members:
ROBERT LINEBARGER, *Brigham Young University,* Provo, Utah
KURT MOSES, *Academy for Educational Development,* Washington, D.C.
H. RAY SOUDER, *Northern Kentucky University,* Highland Heights, Kentucky

This session will deal with the impact of the Information Revolution on our educational systems. It will consider such issues as to what extent this revolution alters our educational mission, whether current corporate and government policies are supportive of this mission, and what future policies will be needed if this mission is to be realized.

Panel: Confidentiality, employment agreement and the mobile employee

Chair:
A. A. J. HOFFMAN, *Independent Consultant,* Forth Worth, Texas
Members:
ROBERT L. GRAHAM, *Jenner and Block,* Chicago, Illinois
ARTHUR E. PARRY, *The Wyatt Company,* Dallas, Texas
JAMES R. STALLARD, *United Telecomputing, Inc.,* Kansas City, Kansas

In our increasingly mobile and entrepreneurial society, companies seeking to protect their trade secrets and other proprietary information must repeatedly contend with departing employees. In turn, employees seeking to expand their employment opportunities are increasingly required to sign confidentiality agreements and employment contracts, which limit both their mobility and their ability to start their own ventures. This panel discusses these developments, with special emphasis on recent legal developments governing employment contracts and confidentiality agreements. Such agreements will be of increasing prominence as employers and employees struggle with the tradeoffs inherent in developing proprietary, competitive advantages in a free society.

Panel: Educational delivery systems—use of networks

Chair:
SYLVIA CHARP, *Past President, AFIPS,* Upper Darby, Pennsylvania
Members:
RICK MACE, *Bell of Pennsylvania,* Bala Cynwyd, Pennsylvania
JOHN PATTERSON, *Temple University,* Philadelphia, Pennsylvania

With the increasing use of networks at the precollege and college levels for both administration and instructional purposes, a need exists to understand better the merging of computing and communication technology. Services to students, faculty, administrators, and researchers can be made available both on and off campus.

Panelists will discuss local networks and metropolitan campus networks and their uses of fiber technology and packet switching. Architectures designed for Carnegie-Mellon University and Temple University will be examined, and applications in school systems will be explored.

Panel: Tutorial on structure editing—developments in modern student programming environments

Chair:
DENNIS R. GOLDENSON, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
Members:
RAVINDER P. CHANDHOK, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
DAVID GARLAN, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
MARK TUCKER, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania

The use of integrated programming environments based on structure editing is an emerging technology that has now reached the stage of being both demonstrably useful and readily implementable. This session contains several related presentations, including an overview of structure editing, mechanisms of implementing programming environments, and their user interfaces. The panelists demonstrate working systems with extensive semantic checking and graphics interface for PASCAL and LAREL THE ROBOT.

Panel: Legal issues in software

Chair:
STEVEN BROWER, *Wood, Lucksinger and Epstein,* Los Angeles, California
Members:
HENRY W. JONES, III, *Vaughan, Phears, Roach, Davis & Murphy,* Atlanta, Georgia
BRENT E. WHOLEBEN, *The University of Texas—El Paso,* El Paso, Texas

This panel discussion addresses current issues in the use and sale of hardware and software. The panel specifically covers the following: The legal implications of providing security, and not providing it, on computer systems; software licensing in Europe; and legal liability issues in educational use. The panel also discusses issues raised by attendees.

END USER COMPUTING

MARVIN EHLERS, Track Chair
Natural Gas Pipeline of America
Lombard, Illinois

Smart buildings for intelligent people

by THOMAS B. CROSS
Cross Information Company
Boulder, Colorado

ABSTRACT

The emergence of the *smart design* and the *intelligent building* will direct building developments in the eighties. Intelligent buildings are structures that include teleconferencing, telecommunications, local area networks, energy management, and other new information technologies. In an increasing number of cases, these buildings are designed by CAD systems.

Today architects are challenged to design buildings that incorporate the new office automation technologies to make people more comfortable, better connected, and ultimately more productive. In designing intelligent buildings, it is necessary to understand new technologies and review the management styles and communications infrastructure of those organizations using the intelligent building.

INTELLIGENT BUILDINGS: INFORMATION TECHNOLOGY AND DESIGN

The emergence of the *smart design* and *intelligent building* will direct building developments in the eighties. Intelligent buildings are structures that include teleconferencing, telecommunications, local area networks, energy management, and other new information technologies. In an increasing number of cases, these buildings are designed by CAD (computer-aided design) systems.

Today architects are challenged to design buildings that incorporate the new office automation technologies so people will be more comfortable, better connected, and ultimately more productive. However, simply introducing smart office equipment into an organization often results in barely-controlled chaos. This phenomenon is clearly evident in many office and building complexes where, for example, a continual rearrangement of equipment, desks, partitions, and people occurs. Because each workplace tends to have a complicated combination of terminals, telephones, and cables, more than one person and one department is usually involved in making the decisions about what and who goes where. Organizations will come to value design features that simplify the automation of offices and buildings. These design changes will run the gamut from good cable management to floor layout.

Some of the many technological and social factors affecting the approach taken to the design of intelligent buildings include the following.

1. Most of the technology which will have the greatest impact on office design over the next 10 years already exists and is in use by leading-edge organizations. How these technologies will be integrated with existing office technology and how rapidly the office worker will assimilate these systems into daily work life will be major issues.
2. Information technology is not easy to incorporate into office building design, but the intelligent building can provide vast numbers of intelligent services without requiring space; power and backup power; heating, ventilation, and air-conditioning (HVAC); and other support systems and management personnel.
3. The specification for intelligent buildings is more stringent than is generally assumed. Office buildings are presently described in terms of the way they are constructed; they are rarely described in terms of the way they are used. Data on how buildings can be designed to incorporate information technology are critical to planning.
4. Currently, office automation manufacturers have addressed only *micro-issues* in the work environment. Innovators will provide a *systems approach.*

In designing intelligent buildings, it is necessary to understand new technologies and to review management styles and the communications infrastructure of organizations using the proposed building.

Though data processing decisions have been monopolized by specialists, this trend is rapidly fading as more and more user departments are making the decision to purchase and control their own systems. Moreover, now that the interface between telecommunications and data processing has been made technically feasible, and people understand that connection, the complexity of decision-making about information technology within the organization has increased. There is controversy about who will manage the new and converging systems, and whether management should be carried out in-house or by systems integrators.

Information technology, especially office automation, is likely to change the way organizations make decisions about the allocation and type of space needed. Studies such as *The Orbit Study,** carried out by Duffy, Eley, Giffone, and Worthington in England, show that a great deal of coordination is required between servicing departments having maintenance responsibilities and the operational departments that procure and use the equipment. The difficulties experienced in installing equipment and distributing terminals illustrate just how interdependent buildings and work operations have become. Consequently, building managers and computer systems personnel often become involved in decisions about future building requirements.

The Orbit Study indicates that a more discriminating client is emerging who will no longer judge buildings solely on traditional criteria (location, rents, finishes, etc.), but on the basis of how well information technology is taken into account throughout the entire building system. Presently, the needs of information technology do not appear to be adequately understood either by suppliers or consumers of office space. Good building performance criteria are sorely needed, and related studies should be a tested. Known rationales for integrating information technology into building designs are as follows.

1. Information technology represents an enormous increase in the levels of capital investment. The hope is that new technology will lead to increased productivity, staff morale, and profitability.
2. There is concern that the office environment be suitable for both in-house and telecommuting employees; specifically, competition for staff knowledgeable of the new

*Published by DEGW, London, U.K. and distributed in the U.S. by Cross Information Company, Boulder, Col. ISBN: 0-923426-05-1.

office technology is increasing (such staff is in short supply in many areas). The quality of the office building environment is likely to influence the choice of jobs and the length of time people stay with a particular employer.

Telecommuting is presently in its infancy, but with changing employee goals it is a high growth area. The office building environment will be altered as a result of employees working at home or on the road. Intelligent buildings should have the capability for connecting telecommuting employees.

3. Pressures from building owners who constantly stress the need to reduce costs and manage building facilities (air, space, energy) more effectively will force the issue of incorporating new technology into building apparatus.

There is a strong need to take the long-range view of the office building and information technology. Certainly other trends will affect development of the various types of intelligent buildings required. Many of the driving forces behind intelligent buildings include the following.

1. The number of small, fast-moving, innovative entrepreneurial firms is likely to increase sharply. Such organizations will adapt to the new information technology more rapidly than older, established organizations and will demand intelligent services as part of the work environment.
2. Home and neighborhood work centers are becoming economically viable. These operations will need high-speed communications links to near or distant headquarters.
3. Large organizations are increasingly delegating decision-making to smaller and more autonomous business units. Improvements in all forms of communication are, in turn, increasing the available options for organizations to either centralize or decentralize their operations. For example, it is becoming possible to centralize administrative services while decentralizing production.
4. There is an accelerating trend among large organizations to cut down main office support and service functions by reducing the number of in-house personnel and contracting out work.
5. There is more pressure on building designers to include more service options such as health centers, video teleconferencing, and even housing.
6. Information technology affects not only the size of office staffs and building requirements, but the ratio of skilled to unskilled employees; it necessitates a different combination of professional, managerial, clerical, and secretarial staff.
7. Greater accessibility to information via computer networking and improved communications will change traditional functional and departmental (space-geographic) boundaries. In the past, office workers needed to be close together in large office buildings. Now they can be located thousands of miles apart and still "feel close" to one another.

Intelligent buildings can also play a major role in the integration of building services. Many organizations simply "bolt on" new technologies, telephones, local area networks, teleconferencing equipment, and satellite receiving dishes; but intelligent buildings require an integrated approach to be successful. The following is a list of major products and services suggested for intelligent buildings.[†]

Intelligent Building Management

1. Online administration and financing
2. Computer-aided design space management
3. Online facilities management and planning

Intelligent Command and Control

1. Life support and comfort—personal comfort environments
2. Online power management
   - uninterruptable power supply systems
   - power generation
3. Wire management—building and local area networks (BAN/LAN)
   - power
   - computer
   - telephone
4. Online building control
   - remote administration
   - simulations
5. Maintenance
   - repair and diagnostics
6. Security
   - access and reporting
   - protection
7. Traffic
   - flow
   - accounting and control

Information Technologies

1. Shared-tenant—telecommunications
   - video teleconferencing
   - audio bridging services
   - audio-graphic presentation equipment and computer-generated slides
   - sale, rent, or lease of telephone station equipment
   - sale, rent, or lease of data communications systems
   - sale, rent, or lease of computer and office systems
   - intercom-only calling service
   - access to local calling service
   - standard and advanced calling features
   - automatic least-cost calling
   - installation of systems

[†] From *Intelligent Buildings and Information Systems*. Report published by Cross Information Company, Boulder, Col. ISBN: 0-923426-03-5.

maintenance of systems
moves and changes
billing and management reports
special account codes
authorization codes
toll restrictions

2. Messaging
   telephone answering
   electronic text messaging and network interface
   voice mail
3. Network interface—carrier and bypass
   access to CATV network/programming
   access to external online databases
   remote access ports
   modem pooling
   packet network interface
4. Paging
   speaker
   pocket pager
   cellular telephone
5. Tenant/text—building directory and information systems (online concierge)
6. Archival storage
   online storage
   vault storage and document shredding
7. Intelligent/movable/programmable office environments
8. Online word/data/info processing—OCR/Facsimile/Telex
9. Tenant support and training (CAI)
10. Encryption—systems security

Intelligent Resources

1. Conference and situation "war" rooms—online room scheduling
2. Temporary services—leased staff
3. Package, courier, and delivery services
4. Travel, conference, and meeting planning services
5. Day care and health/recreational gyms
6. Print/copier/office supplies/services
7. Interior design services (CAD)
8. Educational/training services

CONCLUSION

Intelligent building design stands today where data processing stood 25 years ago. Many knew then what data processing could do, but they did not understand how to do it and did nothing about it. Now is the time to do something about intelligent buildings.

Panel: Smart buildings for intelligent people

Chair:
THOMAS B. CROSS, *Cross Information Company,* Boulder, Colorado
Members:
FRANK DUFFY, *DEGW,* London, England
GORDON LORIG, *AT&T Resource Management,* Basking Ridge, New Jersey

This session discusses the interplay among design, technology, and networks in planning, designing, and implementing intelligent buildings. Issues with respect to local area networks, data communications, integrated micro-mainframe networking, and building management systems are presented.

Panel: Electronic mail—benefits and challenges for the user

Chair:
STEPHEN J. DURHAM, *Cermetek Microelectronics, Inc.,* Sunnyvale, California
Members:
DOUG BRACKBILL, *MCI Mail,* Washington, D.C.
NINA BURNS, *Transend Corporation,* San Jose, California
OWEN GREESON, *Microstuf, Inc.,* Rosewall, Georgia

Electronic mail represents an inexpensive communication system that almost any firm can use and that generally enhances office productivity.

The benefits of such a system are not without challenges. The user must understand how to integrate electronic mail into the office environment. Can existing office data communication equipment work, or must new equipment be installed? Should electronic mail be sent through a store-and-forward database, or should it be sent directly to the user? This session considers these issues and other problems associated with initiating an electronic mail system.

Panel: Managing information centers

Chair:
HENRY DELEVATI, *Information Builders, Inc.,* San Jose, California
Members:
BART BENNE, *Texas Instruments, Inc.,* Richardson, Texas
RANDY CASTO, *United Telecom,* Westwood, Kansas

This session covers areas of interest in today's information center environment. Panelists discuss the concept, yesterday and today, how centers have been run, and past mistakes. The issues addressed include staffing, what skills to look for in IC consultants, user education and training, hardware and software support, user support levels, management strategy, and management acceptance.

Panel: Managing development centers

Chair:
STEVEN GILMAN, *Information Builders, Inc.,* El Segundo, California
Members:
JIM TAVS, *Dean Witter Reynolds, Inc.,* New York, New York
ANNIE WANG, *American Medical International,* Beverly Hills, California

This session explores issues in the emerging fourth-generation-language application development world. It covers topics such as what types of systems should be developed, the size of the systems, the cost, and who pays for them. Key issues involve prototyping methods, implementing standards and procedures, staffing criteria, and differences from traditional development. Among the questions addressed are the benefits and production implementation of prototyped systems.

Panel: Choosing application software—the end user view

Chair:
RICHARD L. LINTING, *Arthur Andersen & Co.,* Chicago, Illinois
Members:
RON ERNST, *Puget Sound Power & Light,* Bellevue, Washington
JOHN HARRIGAN, *State of California Controller's Office,* Sacramento, California
MARCUS HARWOOD, *Hayes Microcomputer Products, Inc.,* Norcross, Georgia

More and more companies are realizing the importance of packaged software, but they are also realizing that it is not a foolproof solution for meeting their information needs. This session helps users and data processing personnel focus on what is required to develop a strategy for the successful use of today's application software packages. Specifically, this session will address two major subjects:

—Essential issues to be considered before selecting a software package. Consideration will be given to the newest technologies incorporated in today's modern application software architectures.

—Key steps in a successful packaged software implementation and some of the frequently encountered pitfalls.

Panel: The micro-mainframe connection

Chair:
CHRISTINE G. HUGHES, *Gartner Group, Inc.,* Stamford, Connecticut
Members:
ROBERT FORTELKA, *Pansophic Systems,* Oak Brook, Illinois
H. F. JONES, *Commonwealth of Virginia,* Richmond, Virginia
DENNIS VOHS, *Management Science America,* Atlanta, Georgia

The proliferation of workstations is real; and with the installed base of PCs at a new level, the market demand is for connectivity. The micro-mainframe link as a promising solution to this connectivity need is discussed from two perspectives, that of the end user and that of the vendor. Panelists discuss a broad range of topics, including a definition of the micro-mainframe market, the problems and benefits of such an implementation, the viability of generic versus vendor-specific solutions, market development issues and their impacts on other products and markets, and what IBM will eventually endorse.

Panel: Managing end user computing

Chair:
WALTER POPPER, *Index Systems,* Cambridge, Massachusetts
Member:
WILLIAM J. SYNWOLDT, *Houston Lighting & Power Company,* Houston, Texas

Most organizations spend enormous time and effort in planning for traditional information systems. Yet they spend very little in planning for end user computing, even though in most companies it is the fastest-growing component of the information systems spectrum. This session addresses management planning techniques for end user computing and office systems. Included are an outline of planning approaches from the perspective of a consultant and an analysis of these techniques from the perspective of a practitioner.

Panel: The ergonomic revolution

Chair:
HOWARD J. MacKENZIE, *Stamco Office Products,* Chicago, Illinois
Members:
STEVEN D. HERRON, *National Safety Council,* Chicago, Illinois
T. J. SPRINGER, *Springer Associates,* St. Charles, Illinois

A comprehensive overview of the issue of ergonomics is presented. The panelists consider how productivity, health, and morale are affected by such diverse factors as the VDT, lighting, sound, visual distractions, and office furniture. The presentation includes results of studies concerning ergonomic problems and commentary on legislation that is beginning to spring up nationwide regarding these issues.

INFORMATION SYSTEMS MANAGEMENT

MARVIN EHLERS, Track Chair
Natural Gas Pipeline of America
Lombard, Illinois

Information systems and competitive advantage

by M. VICTOR JANULAITIS
Positive Support Review, Inc.
Los Angeles, California

ABSTRACT

The dilemma of the last quarter of the twentieth century is integrating information systems technology with a constantly changing business environment. This involves risk, and managers who are measured on a quarterly basis feel that risk is something to be avoided.

Recent history has shown us several cases in which information services technology can play a very important role in a corporation's competitive strategy. The Wizard System, for example helped Avis improve the product and service it delivers to its customers and provide with knowledge about the location, costs, and performance of this fleet. This helped Avis to bargain more effectively with the supplier, and it provided an advantage over Hertz, National, and other rental car firms.

Several distinct types of organizations exist: There are the exploiters of information technology, who change the way their industries do business; the competitors, who use information technology to support their business; and the participants, who only view information technology as a necessary tool.

BACKGROUND

The dilemma of the last quarter of the twentieth century is integrating information services technology with a constantly changing business environment. This involves risk, and managers who are measured on quarter to quarter results feel that risk is something to be avoided.

The questions the Executive poses to us are: "Why should our organization take the risks associated with integrating information services while we are facing critical business issues such as new competition and replacement products and services?" and "What is an acceptable level of risk?" Our response is, "You do it in order to gain competitive advantage through the exploitation of technology." The challenge executives face the last quarter of this century are in meeting this objective.

A competitive advantage exploiting technology rests on developing an information strategy which rests on an understanding of the following five forces:

1. The relative bargaining power of buyers/customers
2. The relative bargaining power of suppliers
3. The rivalry among existing firms
4. The threat of new entrants and
5. The threat of substitute products or services.

Information services technology has helped some organizations to gain competitive advantage in each of these five dimensions.

RECENT HISTORY

Failure to take appropriate risks in exploiting technology can have disastrous results. Take the case of Addressograph Multigraph (most recently AM International). It had a dominant market position in the addressing machine business. During the 1960's however, it did not utilize the new, risk adopting, computer storage and label printing technologies. While AM waited, their competitors, called "List Houses," developed information services techniques to create, store, and print addresses and labels. List houses served the customer better and the market passed AM by. The same thing may be happening today in the financial services industry as companies like Merrill Lynch enter the market and compete with sophisticated information services based cash management systems.

By contrast, there are a number of companies which have expanded their horizons by understanding what the new technology can do for them. We are aware of at least three examples: Avis, American Hospital Supply Corporation, and Whittaker.

Avis

Several years ago Avis, while pursuing a customer service based strategy with a theme "We Try Harder," developed a system which would allow a customer, anywhere in the country, to make an advanced reservation for a car in any city the traveler desired. It was called the Wizard system. To reserve a car with the Wizard system the traveler had only to call a toll free number and was instantly tied into the complete Avis rental network. The traveler mentioned the day, time, and place of the reservation and the computer did the rest. In order to implement this new approach, Avis had to be at the leading edge of information services technology including: remote/high usage/high speed terminals; telecommunications based real time applications code; and, real time operating systems. The risks and costs associated with the Wizard were high. If the system did not work as anticipated, the damage done to the company would be considerable. But the reward of success was a meaningful competitive advantage, so they "bet" their company.

As a result Avis changed the way the travel industry did business. Not only was the development of the Wizard successful, the system allowed Avis to significantly improve its market share. Avis' competition was forced to respond and develop comparable systems. And, while their competition was developing these systems, Avis was able to establish a foothold in an industry in which it previously had no real identity. The Wizard became the standard for entry into the industry.

American Hospital Supply Corporation

In the 1960s executives at American Hospital Supply Corporation (AHSC) developed a strategic plan. They reviewed the distribution function in the health care field and were amazed to discover that there was no single distributor in the field who could service all the supply functions of a hospital. Hospital purchasing agents faced a frustrating multiplicity of products and suppliers. AHSC decided on a full line distribution strategy. Its objective was to make it possible for any hospital or other health care facility to obtain all of their necessary supplies easily from AHSC. However, this strategy could not be implemented effectively with existing manual systems. Consequently they developed a new information services based distribution system called ASAP. A key feature of ASAP was the placement of terminals in each customer's office. Any time the customer needed to order hospital supplies he had merely to key his order into his terminal. AHSC's system accepted it and the material was shipped.

This was not a risk-free strategy. The cost was several millions of dollars. As with Avis this major decision to utilize new technology had a great payback. The success of the system was beyond AHSC's greatest expectations. The company literally drove its competition out of business. Ultimately AHSC expanded this system to support all the organizations under its corporate umbrella. This included manufacturing functions as well as distribution. Today, for example, a hospital in Chicago can place an order for i.v. solution, a manufacturing order is generated at American McGaw's manufacturing facility in Irvine, California, and the shipping and billing functions are all accomplished automatically while volume discounts are being calculated for this hospital's "buying group."

Whittaker

A few years ago, Whittaker Corporation, a holding company, scanned its environment for opportunities for growth in United States markets. It found the hospital supply field potentially highly profitable but highly concentrated. The major barrier to entry was the design of an information-services-based-distribution system to compete with AHSC's. First, a distribution organization was required, so Whittaker acquired a company called General Medical. Second, a computer and communication system like AHSC's was required, which Whittaker had to build. It is apparent that if any company is to compete with AHSC it must do so with an advanced information services system.

These three cases illustrate the central role that information services technology can play in competitive strategy. Information services technology played an important role for each of the firms described above by helping them improve their power position in each of these five dimensions. The Wizard, for example, helped Avis improve the product and service it delivered to its customers and provided it with knowledge about the location, cost, and performance of its fleet. This helped Avis to bargain more effectively with its suppliers. It provided an advantage over Hertz, National, and the other car rental firms. The national network and the service levels it established upped the ante for getting into the business and served as a barrier to entry. In addition to this, the Wizard improved the cost/performance ratios and forestalled the development of substitutes. On the other hand, AM International failed by several of these criteria—especially the inability to forestall substitution.

Technology Exploitation

Distinct categories of organizations exist, some have the ability to exploit technology, others do not.

Exploiters: Information services are an integral part of the organization's unique strategy. These types of organizations have an experience base which allows them to be involved with most leading edge technology and spend significantly more than their nonstrategic technology directed competitors on data processing. One of the interesting characteristics of these types of organizations is the long tenure of the senior management team, including the information services organization. Some industries are technology directed (airlines in the 1960s; national hotel and car rental industries in the 1970s; and the retail and financial services industries in the 1980s).

Competitors: Information services are used to provide the necessary information support for their key strategic business units. These types of organizations have an experience base which will allow them to be involved with one new technology, such as database, at a time. The senior management team does not push its information services group to be first in the application of technology. Rather, it wants to be sure the organization can do anything the competition can do within a reasonable period of time. These organizations typically go through the standard sets of confrontations within their structure establishing priorities. In addition, multiple centers of power and expertise compete for authority in establishing, implementing, and controlling technological direction in the organization.

Participants: Information services are used to provide information for basic management functions such as production, accounting, finance, and marketing. These types of organizations have a very limited experience base and only undertake new technology when forced to. Typically there is one key decision maker who is not in favor of computer or communication technology. One such organization had a president who was proud of the fact that he had "unplugged" all the computers in the organization and it was still profitable. (Given the financial condition of the rest of his industry, he deserved to be proud. He had avoided the technical misadventures which everyone else fell into.) In these types of organizations, if a manager can justify an application of computer technology he is the one who goes to bat for it. The typical application portfolio for this type of organization is focused on operational control types of applications and moving towards the implementation of management control applications. We estimate that between 15% and 20% of corporations fall into this category.

APPROACH TO UNDERSTAND COMPETITIVE ADVANTAGE

How can an organization gain a competitive advantage through information services? We have found that the following five-step process works:

1. Assess the competitive potential of technology within the industry.
2. Measure the risk/change relationship for the organization.
3. Develop the organizational Probability of Success Ratio (PSR) profile.
4. Measure the risk/probability of success ratio for the MIS function.
5. Develop the risk management action steps.

One of the factors which many managers tend to overlook is the current level and dependence on information services technology in their industry. This competitive potential dictates the overall risk any organization in the industry faces from changes caused by technology. The potential is based on two factors:

1. *Dependence:* the depth to which technology is an essential component of the industry. What would happen if the plug was pulled on all information technology in the industry today?
2. *Maturity:* the extent and sophistication with which the industry has adopted the technology. How close to the leading edge of information services technology is the industry?

The combined effect of these two factors reveals the breadth and depth of technology within the industry. A high rating generally implies that the information services in the industry are strategic technology directed. (See Figure 1.)

The three steps to develop an understanding of your industry and organization posture, as depicted in Figure 1, are

1. Identify the industry's major information and communication functions.
2. Rate them as to maturity and dependence.
3. Relate your organization's position relative to the industry.

Then pose questions such as the following:

1. What are the applications of data processing and communication in the industry today and in the future?
2. What is the direction, pace, and momentum of technological change within the industry? The organization?
3. Is the organization behind or ahead of the industry in its application of technology?
4. Are there opportunities to gain a meaningful competitive advantage by leading the industry in information services applications?
5. Can we exploit the technology to support a unique strategic thrust of the organization?

The result is a list of potential technological directions which will provide the organization a leadership position or enable it to gain parity within the industry. Either of these results will entail *changes* in the organization and thus increase its exposure to risk.

A successful organization must achieve a proper balance between growth, control, and technological innovation. The good executive needs to know: "What is the potential bottom line impact of the application of information services technology?" The following five questions are useful in this regard:

1. What is the current strategy for information services?
2. Have the technologies we are using paid off?
3. Do they support the business or drain its resources?
4. How do we compare with our competition?
5. Are we spending the right amount (too much or too little)?

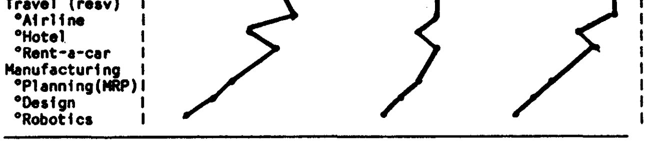


Figure 1—Absorption rate of three industries

Many factors effect the answers to these questions. Included are: the technological dependence and maturity of the industry and the organization; the focus of the organization's application systems—operational control, management control, or decision support systems; the organizational maturity of the computer, communications, user, and management team; the internal performance measurement systems of the organization; and the existing direction, pace, and momentum of implementation.

With all these factors considered, it is a reasonable task to define the set of action steps required for any organization to gain competitive advantage. First, the organization needs to look at the absorption rate and review the systems which are the focus of the future leading competitors. This data can identify the new services a financial services organization is going to provide, or the new products which an office automation company is going to implement, or the new directions which a manufacturing or distribution organization can take to improve productivity. From this, a plan for the information services function can be created. Second, the organization can specify its Risk/Change function and identify the direction, pace and momentum necessary to achieve its plan. Third, by reviewing the individual activities of the information services function, the management team can identify specific action steps required to change its PSR profile and to meet its objectives.

CONCLUSION

Many organizations can achieve a meaningful competitive advantage by developing a strategy based on information services technology. This advantage can be translated into new market opportunities as well as the traditional cost reduction systems. For example, any organization that looks only at a "simple" application of office automation and does not see potential new ways for linking it to its strategy or its basic business units may be missing an opportunity. Companies that have prospered in these difficult times, for the most part, have been innovators. Many of them have innovated in the information services area.

The process presented here is straightforward. The ideas are little more than a new application of good management practices. If your organization is to succeed in the next decade, you will need to manage risk, reward, and probability of success more carefully. These five steps are a way to accomplish this:

1. Assess the technological absorption rate and status of the industry.
2. Measure and plot the risk/change relationship.
3. Develop the organizational Probability of Success Ratio (PSR) profiles.
4. Measure and plot the risk/probability of success ratio.
5. Develop the risk management action steps.

SUGGESTED READINGS

1. Janulaitis, M. Victor. "Gaining Competitive Advantage, or How to Succeed as the Vice President of Information Systems." AFIPS Conference Proceedings Volume 53, 1984, page 513.
2. Janulaitis, M. Victor. "Are the Risks Worth Taking?" *Computerworld,* August 1984, p. 12.
3. Janulaitis, M. Victor, and Richard O. Mason. "Gaining Competitive Advantage—A CEO's Perspective." Malibu, Calif.: Positive Support Review, Inc., 1982.
4. Janulaitis, M. Victor. "The Best of Both Worlds." *Production and Inventory Management Journal,* Third Quarter 1978, p. 1.
5. Nolan, Richard L., and Cyrus F. Gibson. "Managing The Four Stages Of EDP Growth." *Harvard Business Review*, 52 (January–February 1974), p. 76.
6. Keen, Peter, and Michael Scott-Morton. *Decision Support Systems: An Organizational Perspective.* Reading, Mass.: Addison-Wesley, 1978.
7. McFarlan, F. Warren. "Portfolio Approach To Information Systems." *Harvard Business Review,* 59 (September–October 1981), p. 142.
8. McKenney, James L., and F. Warren McFarlan. "The Information Archipelago—Maps and Bridges." *Harvard Business Review,* 60 (September–October 1982), p. 109.
9. McLean, Ephraim R., and John V. Soden. *Strategic Planning for MIS.* New York: John Wiley & Sons, 1977.
10. Porter, Michael. *Competitive Strategy.* New York: Free Press, 1980.
11. Rockart, John F. "Chief Executives Define Their Own Data Needs." *Harvard Business Review,* 57 (March–April 1979), p. 81.

Reengineering business systems

by GEORGE H. RITTERSBACH
Peat, Marwick, Mitchell & Company
Chicago, Illinois

ABSTRACT

The cost of developing replacement business systems is astronomical. The choices of software packages or custom development seem to be multiyear, multimillion-dollar, and multirisk projects. But now the use of software tools can substantially reduce the time and cost of producing target systems through reengineering of in-place applications. Thousands of hours can be salvaged, to a large extent as a result of processing current systems through these structuring software tools. This generates a building-block prototype that can be extended functionally and technically. Users of this approach claim significant improvement in building the application systems they need.

INTRODUCTION

For a generation (a computer generation, that is) software engineers have been building application systems tied to the techniques and methodology of a system development life cycle (SDLC), with the underlying support of a system development project management (SDPM) system. The SDLC methodology is typically phase and task oriented, with the gradual production of a series of specific deliverables, requiring their formalization to be completed around a specific set of forms and documents. The SDPM is used by project managers to monitor the gradual preparation of the deliverables and to manage necessary resources related to the number of application system development personnel assigned to a project. It is also used to track specific tools and capability necessary to support a project and to monitor computer time. SDPM is a recording mechanism that highlights issues, problems, and constraints and constantly reevaluates the estimate to complete the project.

Certainly, the formalized SDLC–SDPM capability has been extremely beneficial in planning, administering, and producing application systems required for major business activity in most corporations. The actual methods and techniques applied to a particular process will produce specific deliverables that can be shared by the project team, presented to users, and become the basis for accepting and ultimately implementing an application system. Nevertheless, these tools present many difficulties to project managers, and executive users for whom the systems are being developed.

Many work activities in systems development projects require participation by several team members, including a user representative, a system analyst, programmers, database professionals, and others. At present, incrementally developed systems deliverables are generally shared by providing copies of paper, which represent the gradual delivery of the system. These often include the definition of data, an analysis of intended work flow, tables of specific codes within the system, and a description of functional requirements. The circulation of all the documents, the continuing reviews, and the updating mean that the "states" of the applications system are hard to keep fixed, and the final preparation of the formalized documentation for the system is often an unpredictable process.

Much of the gradual production of the deliverables requires work to be redone, because increasingly detailed definition typically requires new or replaced forms and documents, and because each participant in a project has a specific idea what the system needs. For instance, the definition of data elements by the user varies significantly from the kind of definition that a programmer desires, and the programmer's definition is different from the more detailed definitions that a database administrator would complete for a data dictionary.

Gaining compliance with the SDLC process is often difficult because of the paper-oriented nature of the activity. The continuing review of the finished work by the project manager (work that frequently is done to a "percentage level") imposes a high project management workload of, simply, administration. Finding, organizing, and evaluating the work done by a project team is frequently difficult and always time consuming. Project participants follow the SDLC–SDPM guidelines to some extent, but almost never, it seems, to the extent necessary to make the project management tasks easier.

The implementation of computer installation standards for systems design, programming, and computer production constraints are difficult to enforce. Considerable investment in work already accomplished on a project, which does not meet defined standards for quality, frequently is overlooked simply because the work is already done.

Toward the end of projects, all of the necessary but uncompleted work related to system documentation, controls, and some functional considerations is reevaluated in terms of the originally planned timetables and targets. If the project is running considerably past intended deliverable dates, the quality of the work drops dramatically in order to get the project done on time.

THE SYSTEM DEVELOPMENT WORKBENCH

The system development workbench is a combination of mainframe computer-resident and personal computer-resident capabilities. It is used as a work station by system development project participants. The workbench capability is provided to assist project participants in the orderly, gradual production of specific deliverables required for the system. Primarily, workbench capabilities are directed at the participants in a project, including the system designer and analyst, the programmer, the users' representative, the database administrator, the data administrator, and the project manager.

A series of specific system development workbench capabilities are included to permit all project participants to produce deliverables gradually and to share their development efforts with all other participants during the course of the project. The workbench also permits the project manager to follow, review, and address the gradual production of deliverables and the timely management and resolution of open design issues, and to address the resource allocation requirements related to meeting schedules.

Project participants make use of the system through a series of work station panels that provide entry into the automated capabilities. From the beginning stages of requirement definition, system design, data analysis and development, work flow and job stream organization, data dictionary development, and database management interface, and on to program code generation, JCL generation, system test planning, system test execution, and other specific tasks, the participants have automated support for their activities. This can significantly reduce the number of persons involved in the deliverable production, and even more significantly reduce the amount of coordination, the number of meetings, and the amount of communication time necessary for such projects. Let us examine the benefits.

Several sets of project team members will generate their deliverables according to predefined processes which are automated and consistent with a given company's standards and guidelines. The deliverables are accessible to all appropriately designated project team members, and passed along from one participant to another for gradual completion as required. The most current state of any deliverable is the only one officially existent; such status is attained only through the project manager's approval. Each project member understands the process, and there can be no misunderstanding, because the automated workbench is a predesignated procedure for systems development.

The reworking necessary under the systems development workbench process is limited because the gradual production of deliverables is in conformance with accepted guidelines, and the work itself is committed to an electronically stored record. Necessary changes are made quickly, officially, and one time only. Immediately, all project participants have the latest version of the deliverable. The project bulletin board—produced once each day—makes note of significant changes in the condition, construct, status, or completeness of any deliverable, so that all project participants are fully informed.

Project administration workloads are significantly reduced by the predefined workbench processes. The benefits of value engineering—doing it right the first time—accrue because it is difficult to forget to do certain aspects of each task or to overlook them completely. Further, the growing volume of system deliverables is organized according to preferred procedure and available to all MIS management, user, project, and other groups as appropriate.

Because the project must follow the automated procedures, and so much of the deliverables are either generated (program code, JCL, testing scenarios) or conform to predetermined constructs, the right work must be done at the right time for the dependent tasks to be started. In essence, the approach of preparing the documentation later, or developing the test plan and data later is not possible without drastically overriding the MIS management prescriptions for the company's systems development process.

BENEFITS OF THE SYSTEMS DEVELOPMENT WORKBENCH

The systems development workbench is among the most exciting and beneficial tools available to application systems developers to date. The integration of productivity aids, code generators, JCL generators, reengineering of current systems, statistical measurement of current systems, predefined testing constructs, CICS generators, code translators (Assembler to COBOL, COBOL to COBOL), and database converters—and many more—within the structure of a given company's SDLC–SDPM and supported by an automated workbench is, simply, awesome.

Consider the major project under consideration at any company that intends to replace automated support that is old, incomplete, and inflexible. Such projects are always multiyear, multimillion-dollar, multirisk and, ultimately, unavoidable. With the workbench, you can:

- Analyze the current in-place applications using PATH-VU to examine the good and bad code in the current systems and for defining which portions can be reengineered and used.
- Analyze the data for the in-place applications to determine problems with its current definition, access methods, database management approach, naming conventions, and the like. Further, you can examine the data for defining static, account, or other classes of code usage and definition, expected values for data names, and anomalies in current usage.
- Prepare the requirements definition and general design of the target system. Using the workbench, the user and systems designer analyst complete this work.
- Estimate the total project scope from a top down view using personal computer-based estimating tools to project ballpark investments for a new system in terms of personnel and computer processing production support.
- Reengineer selected portions of the existing applications using Structured Retrofit to produce state-of-the-art code. This permits the economically sound salvage of staff time in work on current systems, which can be used in the new target applications.
- Design and complete data definitions and structures and implement dictionaries and database managers using the workbench tools for users, database administrators, data administrators, and programmers.
- Generate prototype descriptions of major (or all) inputs and outputs for user assessment and agreement—even testing the use of screens, for instance, leading to final design agreements.
- Generate code automatically using MUTIGEN for teleprocessing support, and several COBOL and fourth-generation languages.
- Generate test plans and scripts, test data, and predefined test results using the workbench tools.
- Manage the gradual production of all these deliverables—and more—through the use of the project manager's workbench.

SUMMARY

The cost of producing application systems for your corporate needs in the late 1980s and into the 1990s is astronomical. That trend has now been reversed. Through the systems development workbench, you can:

- Automate the systems development process through a series of workbench tools that reduce the time and effort of project participants, and increase the quality of the work done.
- Reengineer those portions of current application systems that meet target system needs, and drastically reduce the cost to produce that portion of your target system.
- Analyze data definitions, coding system requirements, and database management approaches through automated assessments of currently defined and used data to reach better defined descriptions of the target systems data requirements.
- Use a series of generation tools to produce COBOL code, 4GL Code, CICS command level code, JCL, testing scripts and scenarios, test data, and more, all for less time and cost than currently experienced.
- Automatically involve users in the systems SDLC–SDPM process through the user's workbench, to better galvanize requirements and deliverables on a basis in which choices can be made productively and with benefit of prototype examples.
- Manage the SDLC–SDPM process using the project manager's workbench to better control the gradual, measured, and reportable application systems building effort.

In a nutshell, with this workbench, you can save time and money, reduce risk, get the quality you want in your systems development process, and produce automated systems that are economically viable for change as your business changes.

Panel: Directions in office automation: An expert's view

Chair:
AMY D. WOHL, *Wohl Associates,* Bala Cynwyd, Pennsylvania
Member:
DALE KUTNICK, *Consultant,* Wayland, Massachusetts

Office automation is no longer a novelty or an interesting experiment for the few. It has become an important element in the successful management of any business. However, it is not easy to make management decisions about what to do in a section of the information processing industry that changes direction so frequently. Even the vendors' names seem to change annually. In this session two office automation experts present their perspectives on where office automation is going and what the data processing manager should do about it. These outspoken experts express their candid views on who's selling, who's buying, and what you should do. An interaction period with the attenders is included.

Panel: Information systems and competitive advantage

Chair:
M. VICTOR JANULAITIS, *Positive Support Review, Inc.,* Los Angeles, California
Members:
MICHAEL S. HESCHEL, *American Hospital Supply Corporation,* McGaw Park, Illinois
RICHARD J. KISLOWSKI, *Denny's Inc.,* La Mirada, California

This session discusses the ways in which organizations can effectively use information systems to the competitive advantage of their companies. Issues to be addressed by the participants will include

—Relative bargaining power over customers and suppliers
—Changing the competitive posture of an organization
—Increasing the cost of entry to new competitors
—Providing new/substitute products and services

Panel: The information systems executive in the 80's

Chair:
JOHN P. SINGLETON, *Security Pacific Automation Company,* Los Angeles, California
Members:
DOUG BOTTS, *Continental Bank,* Chicago, Illinois
EUGENE EIDENBERG, *MCI Telecommunications Corporation,* San Francisco, California

This session focuses on the following topics:
—An overview of major trends for information executives
—Issues in managing distributed data processing and the microcomputer revolution
—Team-building techniques
—Initiatives that management must take to make a better DP executive

Panel: Issues facing MIS executives in the late 80's

Chair:
DALE F. LAKE, *Wickes Companies, Inc.*, Santa Monica, California
Members:
TIMOTHY P. ROCHE, *Beatrice Companies, Inc.*, Los Angeles, California
GEORGE van der VEEN, *Positive Support Review, Inc.*, Los Angeles, California

The increase of automation within the MIS system development process will require the reeducation and retraining of many MIS professionals. This technological advancement will raise many issues that the MIS executive must address. These will include
—Staffing and human resource development
—Organizational and methodological issues
—Information technology itself as an element of change
—Technology changes as a determinant of competitiveness

This panel session considers these and other pertinent issues facing the MIS executive.

Panel: System development workbenches

Chair:
GEORGE RITTERSBACH, *Peat, Marwick, Mitchell & Co.,* Chicago, Illinois
Members:
MILTON JENKINS, *Indiana University,* Bloomington, Indiana
NICHOLAS ZVEGINTZOV, *Software Maintenance News,* Staten Island, New York

The cost of developing replacement business systems is astronomical. The choices of software packages or custom development seem to be multiyear, multimillion-dollar, and multirisk projects. The use of software tools can substantially reduce the cost and time for producing target systems through reengineering of in-place applications. These structuring software tools generate a building-block prototype that can be extended functionally and technically. Users of these approaches claim significant improvement in developing the application dependent systems they require. This session discusses the use of these tools and techniques.

Panel: Office systems and workstations—trends today and tomorrow

Chair:
RANDY J. GOLDFIELD, *The Omni Group, Ltd.,* New York, New York
Members:
JOHN CRAMP, *Merrill Lynch,* New York, New York
MATT GOOBY, *GAF Corporation,* Wayne, New Jersey

This session considers the current technology situation in a large number of corporations today. It assesses the strengths and weaknesses of available technology according to both end users and the individuals responsible for acquisition of equipment within the corporation. After reviewing the current technology environment, the panelists project into the future, predicting trends they see emerging in the next few years.

Panel: Productivity-driven office automation

Chair:
DAVID L. SHAY, *N. Dean Meyer and Associates, Inc.,* Ridgefield, Connecticut
Members:
MICHAEL BLUM, *Peat, Marwick, Mitchell & Co.,* New York, New York
JONATHAN A. BROWN, *GTE Service Corporation,* Stamford, Connecticut
N. DEAN MEYER, *N. Dean Meyer and Associates, Inc.,* Ridgefield, Connecticut

Major office automation projects must be cost-justified to senior management on the basis that white-collar productivity improvement will pay for the technology applied. The panelists discuss phased, business-driven methodologies intended to install quick-hit pilot programs, measure their productivity and quality impact, and project the profit improvement contribution of office automation to the business enterprise.

BUSINESS APPLICATIONS

MARVIN EHLERS, Track Chair
Natural Gas Pipeline of America
Lombard, Illinois

Transaction processing in the reservation industry

by JAMES R. MANCHESTER
Quality International Inc.
Scottsdale, Arizona

ABSTRACT

This paper presents some background on transaction processing relative to the airline and hotel reservation industries. It discusses how hardware and software have changed to be compatible with changes in application priorities. Finally, some of the current stumbling blocks to successful transaction processing systems are identified.

INTRODUCTION

Transaction processing is a term used to describe an on-line, real-time computer system whose prime characteristics are excellent terminal response time, availability, reliability, and recoverability. Each of these terms has specific meaning in this segment of the industry.

1. Response Time: Ninety percent of the responses should occur within 3 seconds or less.
2. Availability: The system should be available in excess of 99% of the scheduled operating hours, which may be 24 hours per day, 7 days per week.
3. Reliability: The system should process messages without errors and with consistent results.
4. Recoverability: The system should recover from most errors without affecting the user or corrupting data or provide alternate means of recovery when this is not possible.

HOW IT ALL BEGAN

Transaction processing in the airline industry was initiated in 1958 by the joint development of a centralized, on-line, real-time reservation system for American Airlines using an IBM 7090 computer and IBM 1301 disk files. *On-line* implies operational and available for use. A good definition of *real-time* is assured, nondeferrable, on-demand servicing of unscheduled, unpredictable, diverse requests in an environment of fixed resources. The intent was to provide both a centralized reservation industry and access to the reservations by agents at the airports, thus giving the passenger the ability to create or modify a reservation at any time from any place. A significant feature in this system was that the name and other pertinent information about the passenger was maintained with the reservation. This also required considerably more storage space for recording the information.

To maintain the Passenger Name Record (PNR) and meet the response time criteria, a specialized reservations program had to be developed, since a general-purpose operating system could not handle the number of messages within the desired response time requirements. This program consisted of a complete operating system as well as an application system tailored to the airline reservations requirements. Pan American, Delta, and American Airlines each tackled the problem in their own way, but jointly with IBM. Each developed a system on a different IBM mainframe. The three systems were all operational by 1964, and performed according to design criteria.

In the sixties, the airlines were growing rapidly. It became obvious that the systems could only handle a specified number of transactions per second before a faster system would be needed. It was also apparent that other airlines would require similar reservations systems. Hence, in 1964, IBM embarked on a new airline reservations system which could be used by any airline. They named the product PARS (Programmed Airline Reservations System). This generic reservation system was designed to operate on the System/360 hardware. It provided a solution that could be purchased by much smaller airlines while providing a growth path for them. Even the largest airline at that time would be able to operate on this family of equipment with the new software.

PARS was the name of the complete package of software that IBM provided the customer. This package consisted of programs performing seat inventory, seat availability, name lists, schedule changes, passenger data entry functions, and programs performing support utilities. There was also a control program bundled in the package, which provided the necessary interface between the applications and the computer itself. This was called the Airline Control Program (ACP).

ACP and PARS met the required performance criteria by providing limited functionality and an architecture based on fixed record sizes and requiring programs to be written in assembler language. There were two record sizes supported for disk files, 381 and 1055 bytes, and a third for main storage use of 128 bytes. This fixed record architecture prevented storage fragmentation and eliminated the requirement for any sophisticated storage management routines. Performance also was enhanced by not editing the application requests to ACP for validity in the on-line environment. Most operating systems check the format and content of parameters provided by an application program. This takes time and is repeated for each request. In ACP, it was assumed that the program was working correctly and that these interfaces had been thoroughly checked out in the testing phase. Hence the editing function was dropped and the machine cycles, which would have been used for editing, were used for performing the requested function.

PARS had not been installed at the first customer location before it became apparent that other businesses had transaction processing needs similar to the airlines and could benefit from the processing speed and functionality of a control program like ACP. Household Finance proceeded to develop an on-line consumer finance system based on ACP. Before long, other businesses such as the car rental, law enforcement, hotel, and credit authorization industries recognized the advantages of ACP and developed their own applications using ACP as the on-line operating system. United Airlines produced such a system for its WESTIN hotels using PARS as a starting point. They took the basic functions of PARS and modified them to support the hotel requirements. For example, a *passenger name record* in PARS became a *guest name record* in this system. This WESTIN reservations system, called WESTRON, is now the basis for the reservations systems used by the larger hotel chains.

Since ACP was being used increasingly in other than the airlines industry, IBM changed the name of ACP to ACP/TPF, where TPF stood for *transaction processing facility,* and began charging for ACP/TPF. It had been available free of charge to IBM customers, but the product is now marketed in all industries where transaction processing is a requirement.

EVOLUTION OF COSTS

Until 1970, computer equipment was extremely costly relative to personnel cost. The emphasis was on writing efficient code to get the most performance from the smallest machine. Programmers would look for ways to save a byte here, a machine cycle there; and as bigger, faster, and cheaper computers became available, programmers became slower, less efficient, and more expensive. The supply and demand process was taking hold.

The design of ACP usually required the customer to purchase two CPUs; one was used to operate ACP, the other was a backup CPU to be used in case the first failed. When everything was functioning normally, the backup was used to provide batch functions supporting the reservations system for management reporting, program development and testing, and certain utility functions. The batch system did not use ACP, and therefore required another organization. Two teams were formed for each function: an operations staff for controlling the computer and a programming staff for the ACP; a duplicate team was also formed for the backup function.

Today the cost emphasis is almost totally reversed from what it was originally. Computer systems which can handle 50 messages per second cost almost a tenth of what they did 15 years ago. Some of the design constraints placed on a programmer have been lifted or relaxed, and the current hardware architecture and speed theoretically make the programmer's job easier.

ACP, unfortunately, has changed little in the manner in which it operates. There are still at least two CPUs required as well as the two supporting sets of personnel. The "user friendliness" of the newer software systems (database management, interactive program development and testing, etc.) has not been added to ACP. As a result, the cost of personnel to support an ACP system is 10% to 25% higher than for other systems. Part of the reason for the lack of change is the huge investment that companies have in their current systems coupled with the original design rules of ACP programming concerning re-entry, linkage conventions, record sizes, etc. Most changes which could be made to the ACP system to take advantage of the newer hardware features would require significant changes to the applications environment. These application changes would have to be performed independently by each ACP user, and IBM is reluctant to permit any user to rewrite their applications.

SUGGESTIONS FOR COST REDUCTION

ACP has had a long history of success in the transaction processing industry and has shown many techniques, such as the use of fixed record sizes, to be valid. The current struggle for performance gains as well as cost reductions point out that ACP is not the perfect solution. From a software point of view, programmers need more tools to reduce the software development time. From a hardware point of view, the standard IBM approach of combining CPUs, either loosely or tightly, does not give linear performance improvements as seen with the airlines and their multiple CPUs in ACP systems.

Hardware Changes

A tightly coupled system or multiprocessing facility provides for the interconnection of CPUs, via a common main storage, in order to enhance system availability and to share data and resources. A loosely coupled system does not have a common main storage but does have the ability to share data external to the CPUs.

A tightly coupled rather than stand alone or loosely coupled system provides a number of benefits. A fault-tolerant system, tightly coupled, is even better. The Synapse N + 1 system has an architecture which is both tightly coupled and fault tolerant. Some of the benefits of this feature combination are as follows:

1. In contrast to ACP, there is only one system to operate, and hence the size of the operations staff is reduced.
2. There is only one operating system to maintain. This reduces the size of the system staff as well as the training requirements of the applications staff.
3. With a tightly coupled, fault tolerant system, single component failures don't necessarily affect the availability or reliability of the system.
4. Increased capacity can be provided at a much smaller increment than adding a new system module (loosely coupled) or upgrading to a faster CPU (stand alone).
5. The Synapse N + 1 system can have equipment added or removed on-line without disruption to the system. When added, the increased capability is utilized immediately. Removal of equipment does not affect transactions which are in progress as far as the end user is concerned.

Software Cost Reductions

Any transaction processing system will experience numerous changes in its lifetime. The software used in its construction should be structured for ease of development and maintenance. This type of software also reduces the development time for most applications. Most structured high level languages, however, are too inefficient in practice (relative to assembler language) to be used in a real-time environment. There currently are two languages that can provide the needed structure and performance: C language and assembler language when combined with structured programming macros and techniques.

Significant improvements in programmer productivity can be had if the system provides a good database manager that also provides speed and flexibility. The programmer is free of database design and integrity concerns and can concentrate on mainline coding. The combination of function and performance is difficult to find in one product, but there are improvements occurring regularly in the industry. Relational databases offer that function. Many of the implementations, however, are not yet up to the performance requirements.

Network/terminal independence is a must for the future growth and change of any system. More and more tools are appearing on the market to enhance this area, but true terminal independence is a long way off. Until then, frequent recoding for new devices will be done.

The potential for a programmer to do one-step programming and documenting is needed. A facility similar to the WEB system of structured documentation, developed by Donald E. Knuth of Stanford University for PASCAL, is needed for other languages. This system provides the ability for the programmer to document the program while coding it. The document portion then is split off from the program by a source analyzer (a computer program) for input into a text processor to produce a final document. The remaining statements are fed to the PASCAL compiler. The programmer maintains only one source file, the tedium of documentation is reduced and the quality of the documentation is improved as well as its ease of maintenance.

SUMMARY

Transaction processing systems have come a long way since the fifties. ACP still is the premier system in the field as far as performance and capacity are concerned. The economics of operating systems today, however, have changed, and ACP has not adjusted accordingly. The marketplace is growing; the demand is there, and new systems are being developed with today's costs and performance factors in mind.

An approach to successful online transaction processing applications

by ARMOND INSELBERG
Synapse Computer Corporation
Milpitas, California

ABSTRACT

Online transaction processing applications are playing an increasingly important role in corporate activities. While transaction processing applications can be characterized as a large number of users concurrently accessing and updating very large databases, there is a fundamental set of generic user requirements which such applications must satisfy to be successful.

This paper shows that with system hardware and software architecture designed specifically to support online transaction processing, the system is able to fully meet these application requirements. Furthermore, by implementing applications on such a system, the impact of these requirements on the development process is minimized; this in turn reduces the time, cost, and complexity of developing and maintaining transaction processing applications.

Following a brief discussion on the nature of online transaction processing, this paper discusses the user's generic transaction processing requirements, the characteristics of online transaction processing application programs, and how a system's hardware and software architecture can be designed to meet the requirements of transaction processing. The SYNAPSE N + 1™ System is used as an example architecture.

INTRODUCTION

There is a major trend to moving applications online. This trend is motivated by corporate competition, customers wanting to know their most current account information, and managers requiring the most up-to-date information in order to effectively make decisions.

In online transaction processing, large databases are concurrently accessed and updated by a large number of online users. Examples of transaction processing applications are provided in Figure 1.

As shown in Figure 2, there are three levels by which the nature of transactions may be defined.

1. A *business transaction* is viewed as the unit of work needed to complete a business function. Booking a customer's airline flight reservation is an example of a business transaction. While making the reservation may include several processing steps and interactions with the system, the reservations clerk considers the activity to be one logical unit of work.
2. An *operator transaction* involves an operator's single request and response interaction with the system.
3. A *database transaction* is a group of database modifications considered to be a single unit. For example, a customer transfer of funds from a checking to a savings account involves several database modifications which comprise a single database transaction. Either all the activities of the transaction must proceed to completion for the database to remain consistent, or none of them are allowed to take place.

The concept of a database transaction emphasizes the fundamental focus of online transaction processing on database activities and database consistency. The definition also provides the foundation for a transaction processing application design model.

Financial
- Online Teller
- Stock Transfer
- Automated Brokerage
- Electronic Funds Transfer
- Automated Teller Machine

Manufacturing/Distribution
- Online Inventory Control
- Integrated Manufacturing
- Just-in-Time

Services
- Wagering Systems
- Videotex
- Information Retrieval
- Reservation Systems
- Hospital Information Systems

Retail
- Point of Sale
- Credit Authorization
- Inventory Management

Figure 1—Transaction processing applications

Hundreds or thousands of users at their terminals directly generate the incoming online transaction processing workload. The fact that transaction processing arises from user-driven workloads results in the following set of generic user requirements.

1. Accurate and consistent database
2. High degree of concurrency
3. Acceptable throughput and response time
4. High application availability
5. Support of unpredictable workload growth
6. Modifiable application functionality and data requirements
7. Secure database
8. Dialogue-oriented user interface
9. Distributed users and integrated systems

These generic user transaction processing requirements are not inherent to the applications themselves; inherent characteristics are such that, if any one of the characteristics is changed, the application is no longer considered to be the same. Since the requirements are not inherent, they should be as transparent as possible during the application design and implementation process.

To achieve transparency, the functionality of the underlying system hardware and software must be designed to support these requirements. Figure 3 is a model of transaction processing showing user requirements, application characteristics, and corresponding required system functionality. These three

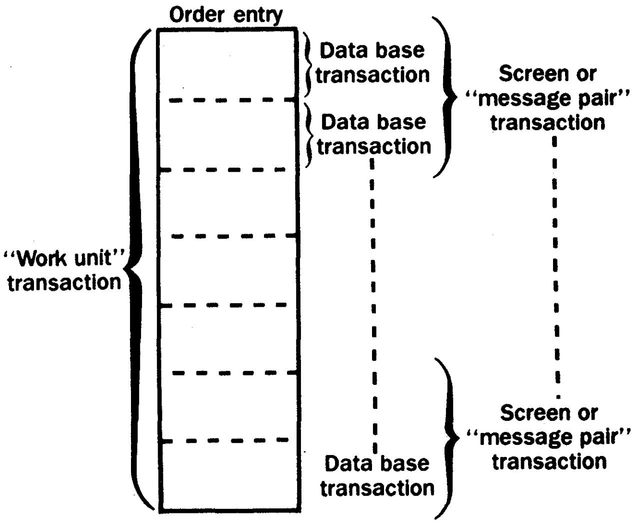


Figure 2—Alternative views of a transaction

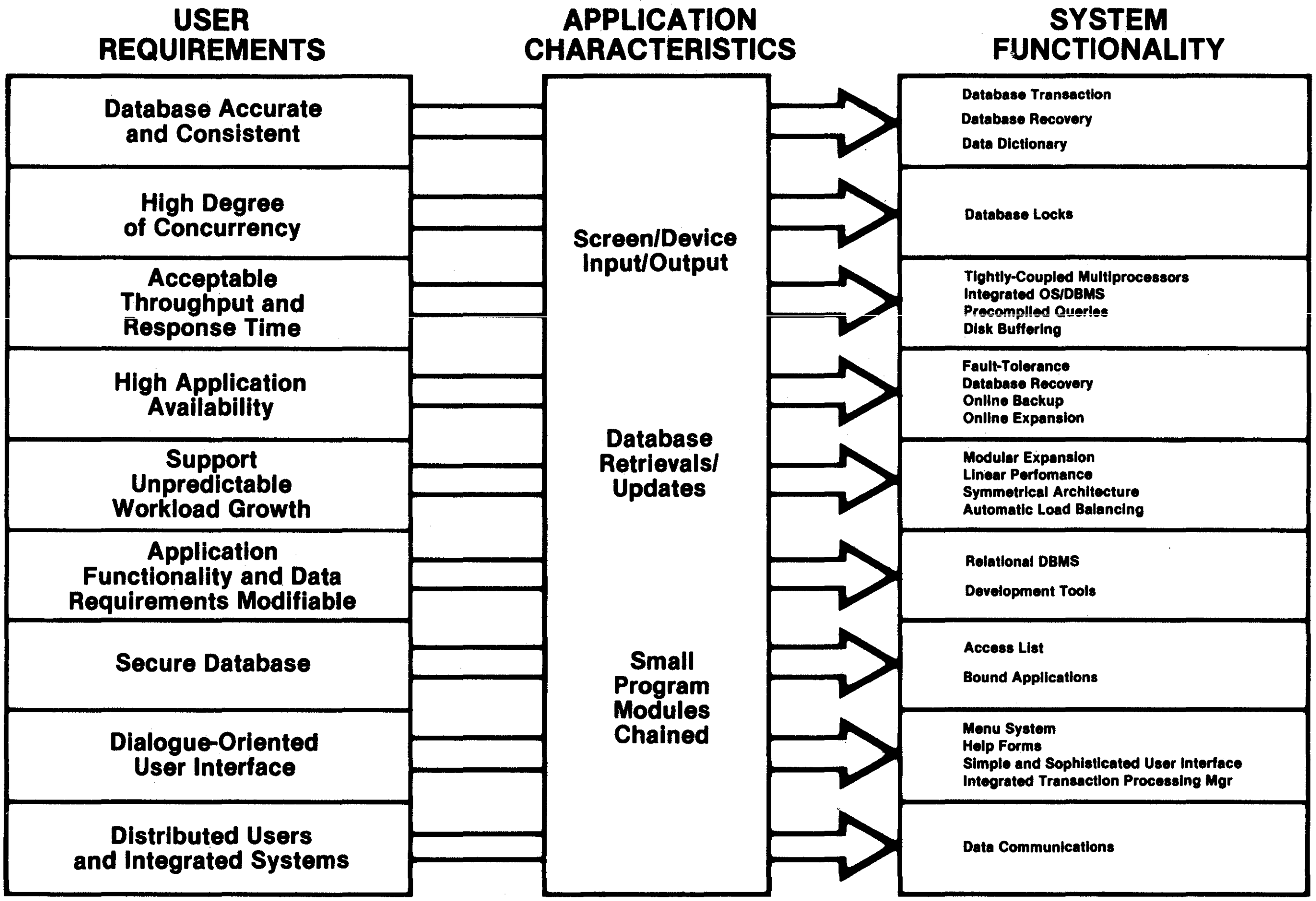

USER REQUIREMENTS
APPLICATION CHARACTERISTICS
SYSTEM FUNCTIONALITY
Database Accurate and Consistent
High Degree of Concurrency
Acceptable Throughput and Response Time
High Application Availability
Support Unpredictable Workload Growth
Application Functionality and Data Requirements Modifiable
Secure Database
Dialogue-Oriented User Interface
Distributed Users and Integrated Systems
Screen/Device Input/Output
Database Retrievals/ Updates
Small Program Modules Chained
Database Transaction
Database Recovery
Data Dictionary
Database Locks
Tightly-Coupled Multiprocessors
Integrated OS/DBMS
Precompiled Queries
Disk Buffering
Fault-Tolerance
Database Recovery
Online Backup
Online Expansion
Modular Expansion
Linear Perfomance
Symmetrical Architecture
Automatic Load Balancing
Relational DBMS
Development Tools
Access List
Bound Applications
Menu System
Help Forms
Simple and Sophisticated User Interface
Integrated Transaction Processing Mgr
Data Communications

Figure 3—A model of transaction processing

aspects of the model are discussed in the next three sections of this paper.

USER REQUIREMENTS

The nine user requirements encompass a wide range of transaction processing applications. These requirements are each examined from the user's viewpoint.

Consistency

It is mandatory that the database remain accurate and consistent, regardless of the type of transaction processing application. To guarantee database accuracy and consistency, only completed transactions are made permanent, and in the event of a system failure, incomplete transactions are backed out of the database. Database accuracy must also be preserved by allowing only valid data to enter the database.

Concurrency

A transaction processing application consists of a large number of users updating a single large shared database. The same set of transactions are executed repeatedly by many users. A high degree of concurrency must be allowed to enable performance requirements to be met. Additionally, synchronization and control must be maintained over the concurrent database accesses to assure that database integrity is maintained.

Performance

Transaction processing systems must meet an acceptable level of performance. Users perceive performance in terms of throughput and response time. While the specific levels of acceptable performance vary widely depending upon the applications, users expect these levels to be met. The defined response time must be maintained throughout the day, including peak transaction rate periods. Consistent response at a particular level is more important than the measured response time for any selected user.

In general, response times of less than three or four seconds are needed to enable a user to remain effective on the job. It is important to note that while users demand rapid response once they submit a transaction, the think-time between transactions is relatively long. Thus, a user's transactions are separated by long periods of reading the last response, thinking about the next data to input, and keying the new input. Batch processing is also needed so that noninteractive transactions can be easily spun-off and processed as background activities.

Availability

As users become increasingly dependent on the online transaction processing application, the significance of downtime increases. *High availability* is often viewed as availability 99.98% of the time. Thus, high availability for a 24-hour/day application requires the total downtime for the year to be no more than 105 minutes. Furthermore, high availability means that the system is not only running, but that the database is accurate and consistent; it does no good for the application to be executing using a corrupt database.

Growth

It is the nature of transaction processing applications that their workloads grow at unpredictable rates over time; the growth may be due to an increase in the number of online users, which in turn increases the number of transactions per second; it may arise from the applications increasing in functionality, thus requiring more computing resource per transaction; or the workloads may grow from the development of new applications. It is important that response times remain consistent as the transaction processing workloads grow. While the functionality of an application tends to vary inversely with the transaction rate, as shown in Figure 4, functionality tends to increase as the complexity of the business environment increases.

Modifiability

Online transaction processing applications must be modified as the business activities that they support change. The modifications may require a change to the functionality of the applications. The application's data requirements may also change, requiring additional data to be held in the database, or the application to share portions of the database previously used by other applications.

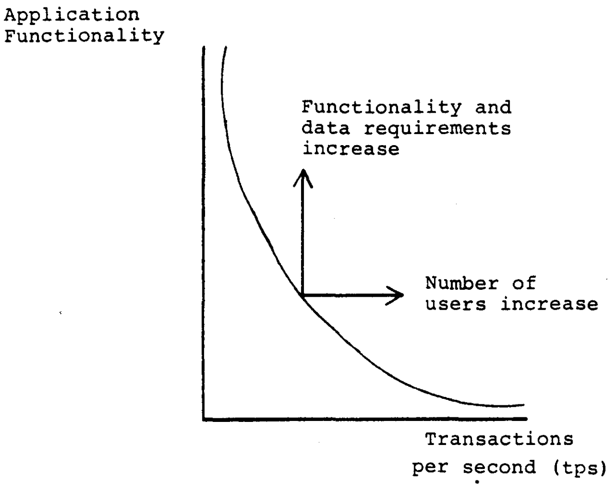


Figure 4—Transaction functionality versus arrival rate

Security

The sensitive nature of transaction processing data requires that the database be secure. Users must be protected from accidental or deliberate attempts at unauthorized access to their data. The security mechanism must provide auditability of such attempts. Furthermore, users must be able to perform both development/testing and production on the same machine at the same time without breaching security.

Dialogue

Offering easy-to-use user interface mechanisms, such as menus and screen forms, facilitates a user's dialogue with the system. The interface should be complemented by an extensive online HELP facility. Such an interface significantly reduces the training requirements and user resistance, especially by users with no data processing background.

Distribution

As online transaction processing applications become broader in scope, it is necessary to address the geographic distribution of users. The users may be geographically distributed throughout a decentralized corporation, and thus need remote access to the central database. As new online applications are introduced into the corporate environment, it is also necessary to be able to integrate these applications into the existing environment.

APPLICATION CHARACTERISTICS

To the user, a transaction processing application executes via a series of menus and screen forms. Such applications generally experience long user think-times, short transaction execution times, repeated use of the transactions, and simultaneous multiple users. In order for applications to fully meet the user's requirements while those requirements remain essentially transparent during the development process, it is important to develop the applications under a design methodology. The methodology takes the form of a design model which defines the structure of transaction processing applications, thereby accommodating the processing needs of both interactive and batch users. The model provides a framework for the design of the applications so that the resulting applications fit the functionality provided by the underlying system software and hardware.

By minimizing the impact of the user requirements during the development process, the time and costs for implementing applications are significantly reduced.

Applications that are designed using the Synapse Design Model[1] are structured, recoverable, and efficient. The pro-

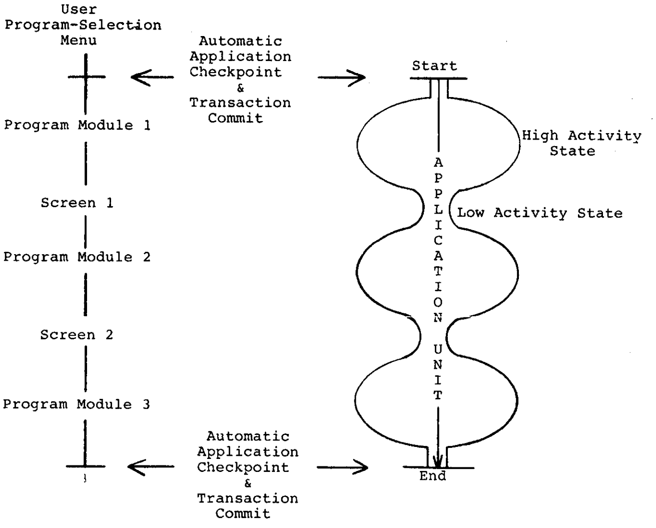


Figure 5—Application activities and resource allocation

grammer simply writes the program modules to indicate which screen forms to be displayed, which data to be selected from the database, and which program module to invoke next. The programmer does not need to be concerned with database recovery, concurrency, and application restartability issues because they are handled automatically by the Synapse system hardware and software.

By breaking the application into small executable program modules, the Synapse model takes advantage of the long think-time typical of the applications. As indicated in Figure 5, by using small modules which invoke each other, the application minimizes the machine resources being used by "thinking" users who are viewing a displayed screen. Resources can therefore be devoted to providing responses to other waiting users. In this way, the system can efficiently multiplex a large number of users over a finite amount of available resources, thereby minimizing queuing and other traditional bottleneck problems.

SYSTEM FUNCTIONALITY

In order for the system hardware and software to fully support the generic user requirements, it is necessary that these requirements be considered at the time the underlying system architecture is designed. Of the three alternative computer architectures available for transaction processing, the monolithic mainframe architecture offers central access to all terminals and the entire database, but with it, entry costs are high, upgrades are expensive and disruptive, and the system is not fault-tolerant. The loosely-coupled multiple computer architecture provides easier upgrades and can be fault-tolerant, but extensive tuning, load balancing and high overhead from interprocess communication reduces its effectiveness. A tightly-coupled multiprocessor architecture with a single shared main memory best fits a transaction processing environment by enabling all terminals direct access to the entire database; system upgrades are non-disruptive, highly flexible, and offer linear performance gains; additionally, the tightly-coupled architecture is fault-tolerant and self-tuning.

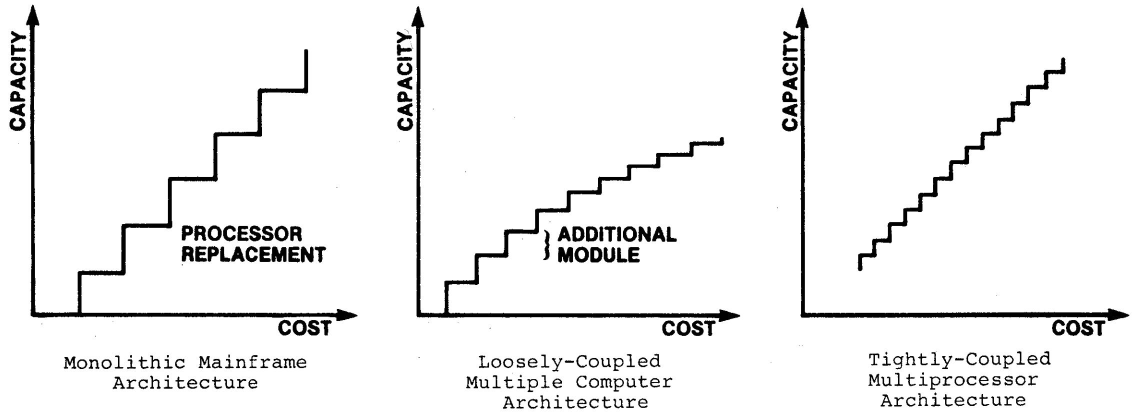
Figure 6 shows capacity/cost growth curves for the three alternative computer architectures.

The following subsections describe how a tightly-coupled multiprocessor architecture, such as the SYNAPSE N + 1 System, effectively meets the nine indicated requirements of a transaction processing environment.

Consistency

The user's requirement for database accuracy and consistency necessitates a highly developed database system logging and recovery mechanism. By considering the unit of recovery to be a database transaction, a database transaction happens either in its entirety or not at all.

As changes are made to the database, they are recorded in a log file. If a system failure occurs, the effects of uncommitted transactions are removed or rolled back from the database, and committed transactions are reapplied or rolled forward. When the recovery is completed, the system will restart the user applications at their appropriate restart point. By this



Monolithic Mainframe Architecture

Loosely-Coupled Multiple Computer Architecture

Tightly-Coupled Multiprocessor Architecture

Figure 6—Capacity growth for major architectures

process, the Synapse system guarantees that no committed data is lost due to a system failure.[2,3]

It is important that the application restart point and committed updates to the database be synchronized; otherwise, following a system recovery, an already committed update may be incorrectly reapplied by the restarted application. Because the Synapse system automatically performs a database commit whenever an application checkpoint is established, this synchronization is assured, and the need for the application programmer to explicitly commit transactions is eliminated. Figure 5 shows that the entry point to and exit point from an application unit are automatic application checkpoints.

Database consistency is enhanced by developing the applications within the context of a centralized data dictionary. All data dependencies are catalogued in the dictionary, thus avoiding data inconsistencies, multiple data definitions, and general application redundancies. The Synapse data dictionary, called the APPLICATION DICTIONARY,™ is fully integrated into the Synapse system and is extenisvely used by the compilers, utilities, operating software, and applications. The APPLICATION DICTIONARY facility plays a major role in protecting the validity of the database. The range, type, and particular value specifications of data fields are used by the system to automatically validate all incoming data from the user's screen. The APPLICATION DICTIONARY also validates all program output to the database on a field-by-field basis.

Concurrency

Having a large number of users concurrently accessing the same database requires that no user see a partially completed update transaction. The system can offer a highly concurrent environment, requiring no additional effort from the application programmer, by automatically locking granules of the database.

In the Synapse system, a granule is a row of a relational database table, or the entire table. This is comparable to record- and file-locking in a non-relational system. Based on the locking granularity specified, when an application accesses the database for update, the referenced data is exclusively locked. Alternatively, multiple read-only locks can be placed on the data. This implicit locking greatly simplifies the concurrency design of the application.

Performance

While the level of acceptable performance varies by type of application, the system must be able to offer a sufficient level of performance to be considered cost-effective. A major gain in performance for a transaction processing system is achieved by using a tightly-coupled multiprocessor architecture. Such a system allows multiple processors to communicate through a single shared memory, thereby significantly reducing interprocess communication overhead. Furthermore, all simultaneously executing processes can independently access the same disks. The Synapse caching strategy not only provides performance gains, but solves the classic multiprocessor bottleneck problems of shared main memory and bus contention. As described later, it is important that each Synapse General Purpose Processor uses non-write-through cache.

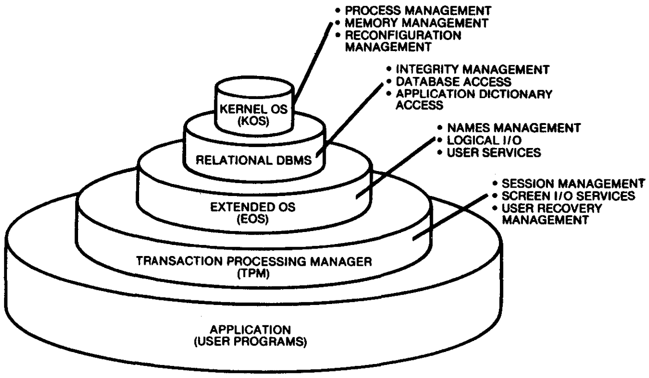


Figure 7—Synapse system software layers

The Synapse shared main memory also allows the use of a single dynamically-allocated disk buffer. For each reference to a database page, the disk buffer is first searched to determine if the page is main memory. If the page is not in the disk buffer, the page is read from disk. The disk buffer reduces the number of disk I/Os for frequently-referenced data. The shared main memory also enables efficient use of system resources by sharing a single re-entrant copy of the system software and application programs. Sharing the executing software in main memory reduces the need to swap programs residing in memory, and enables the system software to be directly available as part of the execution image of each process.

By specifically designing the system software to meet the needs of transaction processing, performance can be further improved. For example, as shown in Figure 7, the Synapse Relational Database Management System (RDBMS) is actually embedded into the operating system software, and the Kernel Operating System is designed to directly support the Synapse RDBMS requirements. This approach is far superior to having the database management system run as a separate process which causes an additional layer of overhead.

In a transaction processing environment, the database queries are predetermined. This enables a large gain in performance to be achieved by determining the database navigation paths prior to runtime. Following compilation of an application, a Synapse utility called the *Transaction Application Builder* determines the optimum way to traverse the database and stores this information in the APPLICATION DICTIONARY as a database navigation path access module. At runtime, the access module is used to accelerate the retrieval process.

Another level of performance gain is achieved by evaluating, at a very low level in the Synapse system software, which rows of a database table satisfy the query. While many systems require that either the application or a high-level portion of the database system determine if the records satisfy the query, performing this evaluation at a lower physical-page level only requires rows satisfying the query criteria to be passed to the application program. This results in fewer exchanges between the application and the Synapse RDBMS, and fewer rows being moved to the application from the database. Further performance is gained by minimizing synchronous writes to the disk for database updates.

Even in an online transaction processing environment, a full set of batch processing functionality must be able to run concurrently with the online applications. Batch activities on the Synapse system may be submitted by a user, or spun-off by online transactions. Batch jobs with similar priority are sent to the appropriate batch queue, assuring the proper work mix and maximum throughput system-wide.

Availability

High application availability means not only that the system is running, but that the database is guaranteed to be accurate and consistent. Therefore, not only must the system be fault tolerant, but it must be able to guarantee database integrity. Fault tolerance means that the system can automatically recover from any hardware or software single point of failure. While applications are not necessarily available during the recovery process, the impact of the system failure is minimized by the system automatically deconfiguring the failed component from the system, reestablishing database consistency, and restarting the applications at their appropriate restart point. Such a recovery process following a system failure ensures data integrity, and thus high application availability.

High availability is also sustained by enabling various operational activities to be performed without requiring the system to be taken offline. Thus, the Synapse system allows database backups and system expansions to be achievable while the system is online. In addition, these operations do not require significant operator effort or expertise.

Growth

Because transaction workload growth is unpredictable, the system must be able to expand modularly as requirements demand. Cost-effective modular expansion is achievable on the Synapse system because processors, channels, primary memory, or various peripheral devices may be added independently. Thus, a tightly-coupled system can expand those specifically needed resources, without having to add costly unrequired hardware facilities.

For modular expansion to proceed smoothly, it is important that the system architecture be symmetrical. This means that all the components must be functionally interchangeable and equitably share the workload. Additionally, a shared main memory enables transactions arriving at the system to be held in a single ready-queue. Because the Synapse General Purpose Processors are symmetrical, they can independently access transactions in the ready-queue. The symmetry of the processors in conjunction with a single transaction queue provides automatic load balancing, thereby eliminating a major ongoing labor-intensive activity by highly skilled technicians.

Another criterion for modular expansion is that the addition of resources provide linear performance. In order to fully expand the system and retain linear performance on a tightly-coupled multiprocessor system, it is necessary that bus and memory contention be minimized. Because each Synapse General Purpose Processor uses non-write-through, private cache, the amount of activity on the SYNAPSE EXPANSION BUS™ caused by a single processor is less than 2% of the total bandwidth of the bus.[4] Non-write-through cache enables changes to be made to a processor's cache without requiring the changes to be immediately reflected in main memory. In conjunction with non-write-through, private cache, the Synapse system uses a proprietary data ownership scheme implemented in hardware to ensure that a processor always obtains the most recent value for each data item.[5]

Another aspect of a system which facilitates its growth is the ease by which system administration can be achieved. By offering straightforward, highly-automated operational facilities, the need for expensive operations staff is eliminated by the Synapse system.

Modifiability

Because online transaction applications are closely tied to the business environment, the applications must be modified as the business activities change. These modifications may require changes to an application's functionality and its data requirements. In the Synapse system, modification of an application's functionality is greatly simplified by the screen forms, database elements, and application programs being defined independently of one another.

The modification of an application's functionality is facilitated through the integration of various development tools. Such tools as the Synapse Program Generator allow application code to be rapidly developed and modified, and user screen forms to be quickly created and altered.

While for many systems the changes to the application's code and screen forms is straightforward, changes to data requirements may be very cumbersome. The difficulty arises when the changes require modification to the underlying database schema. Unless the database system is based on the relational model, modifications to the database schema can require an unload and reload of the entire database under the new schema. The relational model, on the other hand, offers a high degree of data independence from the underlying physical storage structures. Such data independence allows new views of existing data elements and their relationships to be created without requiring modification to the stored database.

Security

Security is important because of the sensitivity of transaction processing databases. Most systems offer front-end security mechanisms to check user logins and OS protection mechanisms which restrict a process to the access of code and data only within its own domain. The Synapse system provides additional security by allowing the restriction of access to system objects, such as files, programs, and batch queues, to various users or groups of users. These restrictions, defined by access lists and capabilities, can allow various modes of access, such as *read* with no *update* capability.

Security in the Synapse transaction processing environment is also provided by the binding of the application's programs, screen forms, and referenced database tables into a single application unit. An application unit provides complete containment of all objects comprising an application. Access to any other tables, forms, or programs is denied by the system. The binding also improves performance by resolving in advance all named references to internal system identifiers, and placing this information in the APPLICATION DICTIONARY.

Dialogue

For the transaction applications to viably support business activities, they must offer a dialogue-oriented user interface. Such an interface must be simple to use, yet sophisticated enough to meet the application's requirements. A hierarchy of menus and screen forms enables the user to utilize the functionality of the application by interactively traversing the hierarchy. Such a menu and forms system is offered on the Synapse system, as shown in Figure 7, by integrating the transaction processing manager software into the operating system. All session threads are managed by the Synapse Transaction Processing Manager, and in the case of a system failure, executing applications are automatically restarted at the appropriate point.

Distribution

To support geographically distributed users, data communications must be in place. Similarly, communication capabilities are necessary to ensure the proper integration of a new application into the existing environment. The integration can often require the interconnection of machines from different vendors. The online system may appear as a satellite to a large backend batch system, with data downloaded and uploaded between the systems. Appropriate means of connecting users and systems include X.25, 3270 terminal support and emulation, and 2780/3780 protocols.

CONCLUSION

The user requirements generic to transaction processing applications must be met in order for the applications to be successful. For these requirements to remain transparent during the development process, and in turn reduce the time, costs and complexity to develop applications, it is necessary for the underlying system hardware and software architecture to be designed to support these requirements. The SYNAPSE N + 1 System exemplifies this approach to providing successful transaction processing applications.

REFERENCES

1. Boctor, W., and K. Cohen. "The Online Transaction Processing Application Model." *Technical Note #5*, Synapse Computer Corporation, 1984.
2. Jones, S. E. "The Synapse Approach to High System and Database Availability." *Database Engineering,* 6-2 (1983), pp. 29–34.
3. Ong, K. "Synapse Approach to Database Recovery." *Proceedings of Principles of Data Base Systems.* New York: Association for Computing Machinery, 1983.
4. Nestle, E., and A. Inselberg. "The Synapse N + 1 System: Architecture Characteristics and Performance Data of a Tightly-coupled Multiprocessor System." *Proceedings of the Twelfth International Symposium on Computer Architecture.* New York: IEEE, 1985.
5. Frank, S., and A. Inselberg. "Synapse Tightly-coupled Multiprocessors: A New Approach to Solve Old Problems." *AFIPS, Proceedings of the National Computer Conference* (Vol. 53), 1984, pp. 41–50.

Front-end transaction processing in the brokerage industry

by ALFRED S. KROIN
Systems Imagination
Jericho, New York

ABSTRACT

Many brokerage systems, such as price-and-quotation display and decision support systems, depend on the timely receipt of pricing information about stocks and other traded commodities. Often these systems subscribe to data services provided by outside sources, and as such are susceptible to reception delays or vendor system malfunctions. This paper presents the alternative approach of user-owned data lines and computer equipment. The relative merits of several systems are discussed, including various aspects of hardware selection and software design.

INTRODUCTION

As is the case with almost every other industry, the past ten years has seen a marked increase in the use of computer systems by the brokerage community. Although these systems have automated widely diversified applications, many share a common basis from which their utility is derived: the prices of marketed securities. Consider the following types of systems: price–quotation display, portfolio management, arbitrage and options analysis, decision support, and historical and archival maintenance. Be they concerned with stocks, bonds, options, commodities, money markets, a combination of these, or another tradable entity, each system depends on the timely receipt of information about the value of its principal items of interest.

In most cases, depending on the degree of timeliness required, the data to drive these systems come from one of two types of external sources. They can be purchased from a "quote vendor" or other communications-based feed (typically a service bureau) if data are needed in real time. If less timely reporting is acceptable, a magnetic tape or other end-of-day service may be purchased. These systems have several disadvantages that exist in varying degrees, depending on the specific data source, including the following:

1. The data are not timely enough. Real-time data may be near real-time, and end-of-day data often are perceived as yesterday's news. On busy trading days, the delivery of data may run behind the actual market position. This can affect the functioning and possibly even the validity of the system.
2. The data may lack flexibility. Depending on the transmission media and the service, the data may not be provided in a form flexible enough for the specific user's needs. Data displayed on a vendor-connected CRT, for example, cannot be relayed to an in-house computer for other calculations or database maintenance.
3. There may be a lack of reliability. Vendor equipment (both on-site and off-site) is beyond the user's control. Regardless of the degree of reliability and fail-safe mechanisms put in place by the user, he is still susceptible to vendor equipment malfunction.
4. Cost factors may be involved. Most services are purchased for monthly fees on an annual basis. The user has no equity and must pay for the services and facilities indefinitely. Fees also are normally proportional to the number of locations and terminals attached; as the system grows, so does the cost of the data.

The alternative to purchasing the data from a middleman is to subscribe directly to one or more of the data communications services offered by the exchanges themselves. There are several widely used services: The Consolidated Tape System (CTS) gives trade prices of the New York and American Stock Exchanges. It is offered by the Securities Industry Automation Corporation (SIAC). The Consolidated Quotation Service/Best Bid and Offer (CQS/BBO) system provides New York and American stock quotations through SIAC. The Options Price Reporting System lists trade and quotation data on stock options as offered by the Options Price Reporting Authority (OPRA). Finally, the NASDAQ Level 1 and National Market Service (NMS) provide quotations and trading information for stocks carried by the National Association of Securities Dealers (NASD).

These services are available to qualified users for relatively moderate monthly fees and can be adapted to almost any application. Although their use does of course require computer programs to be written to process the data, these can usually be provided by in-house personnel or purchased for a one-time fee from outside consultants. Consider how this approach relieves the shortcomings mentioned above.

Because they are purchased directly from the exchange, the data are delivered to the user as quickly as possible with no relay through other facilities. Once the data are captured, the user can process them as he wishes: for display, computation, database maintenance, or any other purpose he desires. Because equipment and site configuration are totally under in-house direction, the degree of fault tolerance is up to the user. Systems at the exchanges typically are highly reliable, but phone lines for the data services usually should be run in pairs through independent switching centers.

Cost benefits are a bit harder to assess because they depend on the user's specific needs. For instance, if an in-house computer is already performing the needed computational functions and it can support the interface for the exchange lines, only the line subscription costs and one-time program development costs will be incurred. As a worst case, if no computer system were in place, the trade-off would probably depend on the monthly charges being incurred and the functionality required. Clearly, if the need exists for any degree of data processing or analysis, in most cases a system will have to be installed. Consideration should be given to integrating exchange data lines as part of the effort. If the data are being used for display purposes only (in the manner of Quotron or Bunker-Ramo), the cost effectiveness of a user-owned system will depend on the number of devices or locations that could be eliminated if a system were available. It is generally unrealistic, however, to expect that all outside devices can be eliminated.

SYSTEM FUNCTIONS AND REQUIREMENTS

Now that we have established some reasonable justification for considering a user-owned system, let us examine the functions the system will have to perform. The requirements imposed on the selection of the computer system itself are shown in Figure 1. Regardless of the ultimate purpose of the system, at a minimum it should perform data capture and processing, database maintenance, and end-user display functions. Each contributes to the requirements of the system.

Although the content and format of the data transmitted by each source vary from exchange to exchange, most services transmit at speeds from 1200 baud to 9600 baud using either a binary synchronous or an asynchronous protocol. Fortunately, these are fairly common and most types of computer hardware readily support their interface. If not, the average user may find it impossible to read the data lines, so users should be sure to select a compatible machine. Most users are knowledgeable enough to be aware of this requirement, but some overlook the potential pitfall inherent to the "free-wheeling" bi-synch protocol. This must be viewed as a water faucet without a shut-off mechanism; the system must be ready to read the data as fast as they are transmitted, or the data will be lost. It is really a one-way line with no facility for the user to request a halt or a retransmission. Naturally there is a recovery capability. The data are grouped into messages and each message is assigned a sequence number. The user thus can keep track of each message he receives. If any numbers are absent, he can call for a retransmission. This is truly a manual process, with a human placing a telephone call to an operator at the exchange site. This upsets the chronology of the data stream and clearly is to be avoided. Although the retransmission facility is certainly needed (no system is totally foolproof and communications lines occasionally contain noise), the user should select a computer system with sufficient capability and capacity to ensure that messages are not missed during normal operation. Such a system is said to be able to keep up with the lines.

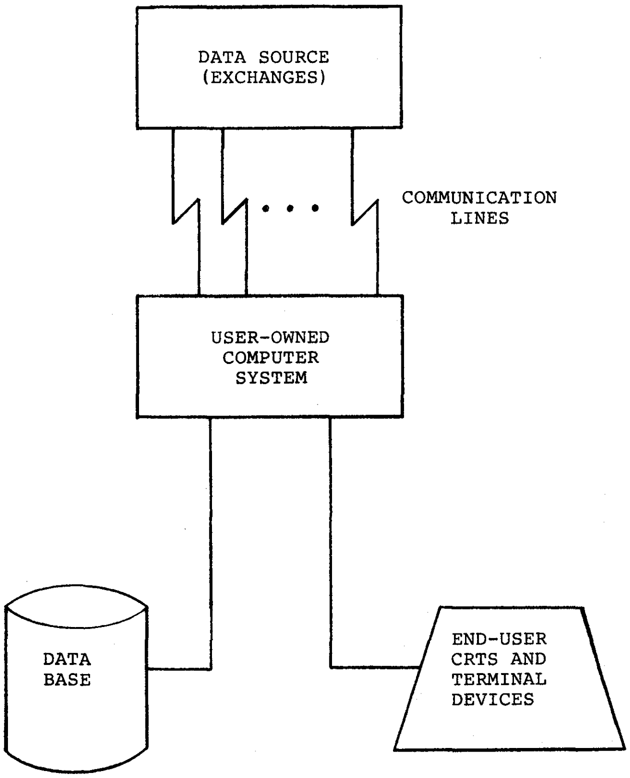

DATA SOURCE (EXCHANGES)
COMMUNICATION LINES
USER-OWNED COMPUTER SYSTEM
DATA BASE
END-USER CRTS AND TERMINAL DEVICES

Figure 1—Components of a front-end brokerage system

By examining the components of a typical transaction, it is possible to evaluate system requirements both for capability (what needs to be done) and capacity (how much). Each transmission is composed of a minimum of three components. They are data capture, data processing, and database update or computation. The data capture function is probably the most complicated, particularly to those who are unfamiliar with data communications. The less familiar a user is with this technology, the more careful should be the selection of a computer system to support the protocols needed. A system that incorporates vendor-supplied hardware and software to read the data lines on behalf of the user is ideal, because it greatly simplifies the user interface. Systems manufactured by Synapse Computer Corporation and Tandem Computers do this quite nicely, although Synapse conveniently puts more of the processing in its communications controller, thus reducing the overhead at the central processing unit.

Without such features, the user may suddenly find himself writing code to search for control and other framing characters (SYNs, SOHs, and ETXs) and performing parity check computations. The best way to evaluate a prospective vendor's ability in this area is to request a benchmark to enable evaluation of the level of effort required to create code to read the lines. If you are fortunate enough to have access to the specific exchange services under consideration, you also will be able to assess processing requirements and the number of transactions that can be captured per CPU-second.

Although it probably requires more actual lines of code than any other basic function, processing the data read is surely the most straightforward. Each exchange service provides accurate documentation on message formats. It is the parsing of these messages that we refer to here as data processing. User-specific processing beyond this—and its effect on system requirements—must be treated on a case-by-case basis. If the user facility has no programming personnel, software packages are available for the most common services, or consultants can be hired for customized efforts. Processing power used by this function is also minimal in comparison with that needed by the other components because no input or output operations are performed.

In most systems, the next operation to be completed will be some form of database maintenance, usually to a disk drive or other mass storage device. Two exceptions to this are wholly main memory resident systems or purely event-driven decision support systems. Because it is not unusual for a database to contain more than 5 megabytes, those systems that are wholly or predominantly memory resident typically monitor only a subset of the securities carried by the line services, or they may use a hybrid memory–disk approach. They rely on data maintained, processed, or retrieved from main memory and do little or none of the I/O operations that bog down systems and consume CPU power.

Great care must be taken with these applications, however, to protect the data in the memory from system malfunctions. It is usually recommended that duplicate copies of the data be kept in the memory of separate processor modules to facilitate recovery from malfunction or data corruption. Unfortunately, this is costly; large amounts of memory, processor modules, and disaster-recovery hardware often are necessary to ensure system reliàbility and data integrity. Purely event-driven decision support systems sometimes perform no database maintenance at all; they merely act on the data as they are processed and then discard them. Here the system burden depends heavily on the particular application and only by precise computation or benchmark can the hardware requirements, capacity, and response time be estimated.

The vast majority of systems, however, do employ disk-resident databases. Many of them in fact perform several file or record updates for each transaction. Let us assume, for the purposes of our analysis, however, that only one database update is required per transaction and that an update consists of two I/O operations, a read (to retrieve the record of interest) and a write (to revise it on disk). The reader can extrapolate the analysis presented here to a particular situation. Some systems, such as the Synapse, employ sophisticated global memory disk cache schemes to reduce the number of I/O operations. Physical database updates occur at user-defined consistency points rather than one per logical update, as is the case with most other systems. Because this can yield an improvement in database management, it is a significant feature to look for in selecting the hardware. This becomes quite clear when a closer look is taken at the numbers involved.

A single, high-speed line operating at 9600 baud transmits 960 characters per second. Using an example of 1000 characters per second (to make the numbers easier) and an average message length of 40 characters, this yields 25 messages per second. If, as with conventional systems, this translates to 50 physical I/Os per second, the amount of hardware required for each data line becomes prohibitively expensive. That is, if a nominal disk operation requires 30 milliseconds' average access time and about 100 ms of CPU time, 50 transactions will use 1.5 disk-seconds and 5 CPU-seconds. Consequently, we would need at least two logical disk drives (actually, four, to allow for mirroring) and five processor modules for each exchange data line desired (exclusive of other processing requirements). Without such features to improve the I/O efficiency, a user may not only spend money unnecessarily, but also may be unable to configure a system large enough for the application (due to coupling restrictions of multiprocessor architectures) or may outgrow it prematurely (by not being able to keep up with expansion requirements and needed enhancements).

DESIGN ASPECTS AND SOFTWARE CONSIDERATIONS

We have reviewed briefly the realm of brokerage applications and discussed some of the features affecting the selection of their computer systems and performance. Let us now direct our attention to some design aspects and other software considerations. Any system of the 1980s must embody the one feature that is the computer industry's equivalent of flag and motherhood: modularity. There are many parallels in logic from one exchange service to another and a high percentage of the code written can be used as a skeleton for other services if a system is modular in design. Modularity improves the development cycle from the standpoint of both time and cost.

One design has proved effective in several applications. The basic process architecture (shown for a single data service in Figure 2) can be replicated for multiple exchange lines as the hardware permits. Exclusive of application-specific code, the system is composed of three different processes interacting in requestor–server relationships to perform communications, transmission, and message level processing.

Communications Handler

The communications handler executes all I/O operations to read the raw data from the communications controller. It

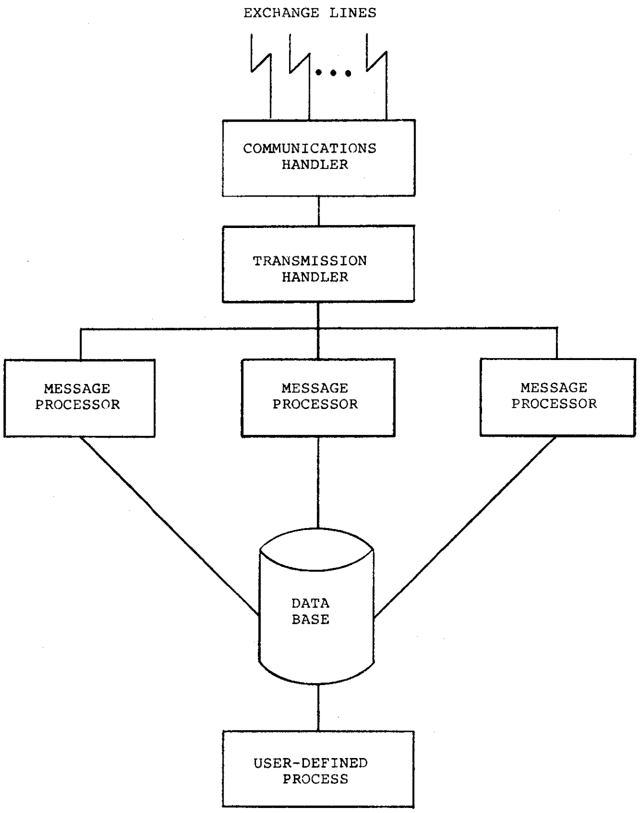


Figure 2—Basic process architecture

provides maintenance of all protocol requirements, does block level format validation and error reporting, and accumulates block level statistics.

Transmission Handler

The transmission handler parses blocked data into individual messages. It checks message sequence numbers, validates message parameters, collects transmission level statistics, and reports any errors in message level header format and content.

Message Processor

The message processor determines message interest and further processing as required, validates message content, and revises and creates—if necessary—pertinent database items (for pricing, volume, and other security data). It can also relay the data via an interprocess queue to a user-defined process (e.g., a computation process or a process for handling a local area network).

Because (in the Synapse implementation) the actual communications line driver and the majority of the protocol management is accomplished by a separate, dedicated processor external to the CPU, all buffering of transmissions is done by the line driver itself. The communications handler simply posts a read, receives and time-stamps each transmission, and sends it on via an interprocess communications (IPC) facility (actually a message queue) to the transmission handler. In the Synapse system, interprocess message acknowledgment is asynchronous to message dispatch. That is, one process can continually send messages to another without waiting for a reply. The operating system manages the interprocess message queues in memory, automatically mapping them to contiguous disk space as they grow. Some earlier approaches on other hardware required buffering of messages on disk, which increased both physical disk I/O and program complexity.

The segregation of functions between the transmission handler and message processor is probably most significant, however. Because the former deblocks messages and performs all message synchronization and control, the message processor's only function is to complete the decision logic of data extraction and database access. As such, this process can be "cloned" as necessary to keep up with data reception, forming a many-server-to-one-requestor relationship with the transmission handler.

To maintain consistency between the messages processed by the transmission handler and those the message processors must apply to the database, a separate relational database table is used. Each message processor owns a corresponding row in the table where it stores the sequence number (as sent by the exchange) of the last message processed. The transmission handler distributes messages (via IPCs) to each message processor, based on a modulus of the sequence number. This provides an easily traceable series of messages processed by each message processor. In the event of a systemwide failure, the transmission handler simply scans the sequence number table to determine the last message sent to each message processor. This implementation of sequence number synchronization should be noted both for its simplicity (using standard database procedures) and its flexibility (allowing multiple copies of the database-intensive message processor). Also, although it depends on the vendor's database system for recovery, this method is as fast as a memory-resident array because the sequence number table is in the global memory disk cache and accessible to all running processes. That is, in the Synapse system, all main memory can be shared among all processor modules.

In this approach, one other process needed to be added before tailoring to the application's requirements. An error handler was created to receive IPC messages from all other processes. It logs and time-stamps all missing sequence number reports, retransmission requests, and system errors to a designated terminal or hardcopy device, while maintaining a disk file log as well. This relieves the other processes of responsibility for logging errors and provides a central system-monitoring capability. Another important aspect is the benefit inherently derived from Synapse's relational database management system. All database concurrency considerations, including record locking, deadlock detection, and recovery, are handled by the DBMS and not by the programs. This speeds system development and reduces complexity at the user level.

Time and space do not allow examination of all of the branches a system based on this architecture could follow, but several points should be made regarding the way the design might proceed and some miscellaneous features to seek out in the host system. If the system is to be used solely for local data display, reporting, or other retrieval functions, the only development left for the user is to provide programs for database access. These are usually asynchronous or batch operations in relation to the primary database maintenance function. It should be simply a matter of programming to complete the effort. The ease of development rests with the skills of the programmer and the quality of software tools provided by the computer vendor. Users should therefore carefully evaluate each prospective vendor from this standpoint. Other candidate user applications, either computational (arbitrage or decision support systems) or I/O-oriented (distribution through local area networks), will require more effort to complete, but they are similar enough in the next stage to be considered together. Either can be treated as a fourth-level process below the message processor, using the same IPC capability to pass each transaction to the application. Again, the ability of the host system to execute this function and support a modular design to replicate processes at the user level is essential. If the user keeps in mind these features—and the other suggestions mentioned—in selecting any of the computer systems that are highly reliable and expandable, he can be assured of having the foundation of a successful system.

Panel: Business graphics—the experts predict the future

Chair:
ALAN PALLER, *AUI Data Graphics/ISSCO,* Arlington, Virginia
Members:
NEIL KLEINMAN, *International Data Corporation,* Santa Monica, California
ANDERS VINBERG, *ISSCO,* San Diego, California

The focus of this panel is short- and long-term trends in computer graphics use in management. Equipment trends, including the impact of workstations, is discussed. Trends in software, applications, and management are considered. Specifically, software trends include windows, artificial intelligence, and the new one-button user interface for executive access. Applications trends encompass such topics as production graphics, scientific charting systems, and visual early-warning systems. Finally, management issues include planning for complete visual information systems and for the impact of graphics on the relationship between information system departments and top management.

Panel: Optical and video technologies' delivery systems

Chair:
D'ELLEN BARDES, *Alltech Communications,* Denver, Colorado
Members:
M. SCOTT ALBERT, *Criterion Venture Partners,* Houston, Texas
J. OLIN CAMPBELL, *WICAT Systems,* Orem, Utah
PETER CROWELL, *The Advanced Learning Systems Organization,* Blackhawk, Colorado
EDWARD S. ROTHCHILD, *Rothchild Consultants,* San Francisco, California

Available optical disk and image technologies permit the integration of high-quality pictures with existing text/data/graphics management systems. By providing users access to and manipulation of visual files, innovative archival, merchandising, and management applications are made possible.

Which of the overlapping, cost-effective peripherals or integrated solutions should one select—analog videodisk, optical digital data disk, high color-resolution graphics boards, or compact disks? Industry experts discuss alternative visual information systems' advantages and disadvantages, industry players, applications, and implementation concerns.

Panel: Computers in manufacturing

Chair:
WILLIAM G. RANKIN, *Deere & Company,* Moline, Illinois
Members:
ROGER M. BURKHART, *Deere & Company,* Moline, Illinois
DAVID C. SCOTT, *Deere & Company,* Moline, Illinois

Deere & Company is recognized as a leader in the application of computer technology to manufacturing. This session includes presentations covering a brief history and background, present activities, and future directions. Applications described range from analytical tools used in planning and designing manufacturing processes to direct control of automated devices on the shop floor. Particular emphasis is given to rapidly advancing computer technology and its integration into manufacturing operations.

Panel: Decision support systems—the end user view

Chair:
EPHRAIM McLEAN, *University of California at Los Angeles,* Los Angeles, California
Members:
FRED BRACHMAN, *Brachman Associates,* Pennfield, New York
C. LAWRENCE MEADOR, *Massachusetts Institute of Technology,* Cambridge, Massachusetts
DAVID NESS, *University of Pennsylvania,* Philadelphia, Pennsylvania

The participants in this panel session have all had extensive experience in DSS. Ness, who has worked on DSS at the Wharton School, presents observations on the status of the field and where he sees it going. Meader, who has recently completed a major study of several companies using DSS, reports on these results. Finally, Brachman describes his experience with DSS at Kodak and the successful applications they have achieved.

Panel: Current market applications for smart cards, keys, and tags

Chair:
LAWRENCE R. KILTY, *The Kilty Company,* Bethesda, Maryland
Members:
ROY BRIGHT, *Intelmatique,* Paris, France
SAM EPSTEIN, *Analytics Communications Systems,* Reston, Virginia
BILL FLIES, *Datakey, Inc.,* Burnsville, Minnesota
ROBERT A. KITCHENER, *Casio Microcard Corporation,* Armonk, New York

This session focuses on actual market applications for smart cards, keys, and tags that are in operation and available to manufacturers and OEMs alike. These portable information devices are being used in a host of areas. The panelists present a current overview of four major application areas:

—The Electronic Dog Tag system: Process control applications; medical and personal history systems
—Communications security applications
—Applications in the credit and debit card industry
—Use in telephone, teletext, and point-of-sale applications

Panel: Security—now and in the future

Chair:
WILLIAM BRAMER, *Arthur Andersen & Co.,* Los Angeles, California
Members:
RUSSEL W. JENKINS, *Wells Fargo Bank, N.A.,* San Francisco, California
MORGAN MORRISON, *Glendale Federal Savings and Loan Association,* Glendale, California
CLIFFORD A. MORTON, *Boise Cascade Corporation,* Boise, Idaho

In spite of the wide variety of security management tools and techniques available today, most organizations find that gaps remain in their defense. The dynamics inherent in information technologies ensure that this will always be the case. This session reviews today's security defenses and the frequent shortfalls in trying to achieve their full benefit. The participants propose steps an organization can take now to be better positioned for tomorrow's changes.

Panel: Output devices

Chair:
ALAN SOBEL, *Lucitron, Inc.,* Northbrook, Illinois
Members:
CARL MACHOVER, *Machover Associates,* White Plains, New York
ROBERT A. MYERS, *IBM T. J. Watson Research Center,* Yorktown Heights, New York
MARK TABAK, *Xerox Electronic Publishing Unit,* San Diego, California

The quality of the output from a computer—"soft," like a display, or "hard," like a printout—can have major effects on the way people use the output. CRTs produce most soft copies, but flat panels are beginning to appear in serious numbers. Similarly, the dominant impact types of hard-copy devices are meeting increasing competition from non-impact printers. All four of these general types of output devices are described and projected advances in the technologies discussed.

PERSONAL COMPUTING

JAMES GERDES, Track Chair
Argonne National Laboratory
Argonne, Illinois

Controlling third-party testing vendors

by RUSSELL R. SPRUNGER
Graphic Software Systems
Wilsonville, Oregon

ABSTRACT

In today's fast-paced microcomputer world, getting a high-quality product to market in a timely fashion can make the difference between long-term success and mere survival for a company. Independent laboratories can provide valuable testing, easy mechanisms for managing project costs, and access to experts not available in a developing organization. This paper outlines the benefits of using independent testing organizations, proposes a method for selecting such an organization, and describes effective mechanisms for their use.

INTRODUCTION

In today's fast-paced microcomputer world, getting a high-quality product to market in a timely fashion can make the difference between long-term success and mere survival for a company. How to test products adequately is a crucial consideration in the development process. Independent hardware-testing labs have existed for many years, but independent labs to test software have become a reality only in the past few years. Independent laboratories can offer professional services, enable effective project cost management, and provide access to experts who would not normally be available to developing organizations. This paper outlines the benefits of using outside testing organizations and describes effective use of their resources.

BACKGROUND

In the summer of 1983, Graphic Software Systems (GSS) was preparing for the fall introduction of a series of new graphic software products. At that time, GSS had devoted most of its resources to developing software products and had not established a formal testing organization. The challenge was to develop and implement a plan for testing the new products. The testing had to be prompt, comprehensive, inexpensive, and reliable under repetitive use.

At the base of the product series to be tested was GSS-DRIVERS, a device-independent graphics extension to MS-DOS and Unix. Therefore, the proposed tests had to exercise a wide variety of functions that take into account the device-dependent attributes of each device. In addition, this variety of test functions had to address the devices then supported by GSS as well as software developed in the future to support additional graphics devices. Because GSS products are designed to be usable on different operating systems, the tests also had to be portable between operating systems.

Having outlined the goals and constraints for software testing, GSS began to investigate ways to implement the tests. One possibility was to hire full-time employees, thereby establishing the foundation for a quality assurance group within the company. Other possibilities included using qualified customers or outside contractors.

Finding qualified people to staff a new department immediately was a task that presented more difficulties than advantages. Using qualified customers had provided valuable feedback, but historically had failed to result in comprehensive, well-documented, portable testing. Therefore, GSS began searching for an established group that could satisfy our requirements. We began our search with several organizations referred to us by our colleagues.

EVALUATING TESTING VENDORS

The next step was to determine how to qualify and select the right organization. Initial qualification consisted of contacting several organizations and requesting company background, specialization, and general business terms. This process resulted in two organizations who advertised product testing or installation. Each had some experience in testing graphics products.

Next, each organization was visited to evaluate the personnel, previous work, and the work environment itself. This activity also involved discussing the product to be tested and providing the test organizations with enough product material to allow them to provide GSS with a formal quote.

DEVELOPING THE FORMAL QUOTE

From this evaluation process, we determined that the formal quote should contain at least the following items:

1. Detailed description of work
   a. Overview
   b. Number of tests
   c. Language of implementation
   d. Test methodology
2. Schedule
   a. Start and end date of project
   b. Deliverables by developer by date
   c. Deliverables by contractor by date
   d. Number and location of meetings
3. Costs
   a. Personnel
   b. Travel
   c. Equipment (hardware and software)
4. Business terms
   a. Payment terms
   b. Ownership of deliverables

Following on-site evaluation and receipt of a quote, GSS selected the International Bureau of Software Test (IBST) of Sunnyvale, CA.

MANAGING THE TESTING VENDOR

Once the contract had been awarded, technical and management representatives were designated and regular meetings were scheduled. Weekly meetings were determined appropriate for projects of up to three months duration. Owing to the distance between GSS and IBST (about 600 miles), many of these meetings were conducted over the phone. On-site visits to the test location were conducted every three weeks.

As with any testing effort, it was important to define an effective means of addressing each product error and of informing the testing organization about new releases of the product. IBST provided weekly reports in addition to the meetings. The reports included project status and detailed software error accounting. Detailing and then following up on each error has been addressed in subsequent releases of software. This was a significant task, both in its importance and in the resources expended.

At the end of the project, GSS had a complete, documented set of tests that could be run regularly as new installations occurred.

BENEFITS OF INDEPENDENT TESTING

The use of outside testing resource has revealed several benefits:

1. Easy management of the highly variable demand for human resources
2. Well documented testing of first-time user situation
3. Complete, documented test suites
4. Good user interface feedback
5. Verification of product conformance to stated documentation
6. Verification of product conformance to standards

The earlier in the product's development that the testing organization can be involved, the greater the potential for finding design errors before they become ingrained product deficiencies. Simple economics dictates that correcting problems at the design stage is much less expensive than having to rebuild the product, because testing is relegated to the final stages of product development.

An additional advantage of earlier involvement of the testing organization is that the laboratory will have time to learn about the product, thus enabling them to be more expansive in the design of their procedures. This goes beyond simply ensuring that the product conforms to the primary level of usage suggested by the documentation. Providing the organization with the opportunity to perform in-depth testing permits optimization of the testing process.

STEPS FOR CONTROLLING TESTING VENDORS

The following steps outline the process of using outside organizations to perform product testing:

1. Designate a project leader
2. Determine what you hope to obtain by using an outside organization
3. Locate available outside testing organizations
4. Determine the business procedures of these organizations
5. Specify what you need to have tested
6. Obtain quotes for services
7. Evaluate test proposals on the basis of
   a. Cost
   b. Staff applied to task
   c. Types of testing performed (validation, stress, destructive, ease of learning, ease of use)
   d. Length of test cycle
   e. Amount of retest
   f. Test deliverables
   g. Completeness of test cases
8. Negotiate the contract
9. Begin the contract
10. Monitor the project
    a. Weekly for multimonth projects
    b. Monthly for multiyear projects
    c. Evaluate test procedures
    d. Acceptance of project deliverables
    e. Face-to-face vs. long-distance monitoring (face-to-face preferred at least 4 times during contract)
    f. Start test acceptance, following each major deliverable
    g. Final presentation of test findings
11. Study test deliverables

SUMMARY

Outside test organizations can provide many benefits. To use these valuable services effectively, the product-developing organization should assess its testing needs accurately, involve the testing organization as early as is practical, and demonstrate proficient project management skills—particularly the ability to manage projects long distance.

Software testing procedures

by LEE SPRAGUE
The International Bureau of Software Test, Inc.
Sunnyvale, California

ABSTRACT

Although perfect software does not exist, the use of thorough testing procedures greatly reduces the number of possible errors. The earlier testing procedures are introduced, the greater the developer's cost savings.

Independent software testing laboratories are working with standard-setting organizations to establish and define testing standards. Such laboratories provide objective, accurate, and economical testing through detailed testing plans and procedures that ensure that all items are tested and that tests can be repeated. Testing plans identify testing levels, types, methods, and procedures.

Levels of testing include unit, system, and integration. Types of testing include functional, destructive, regression, and documentation. Testing methods include procedural, outline, ad hoc, comparison, and compatibility. Software testing procedures consist of manual inputs, semiautomatic inputs, or automatic inputs and may require visual review of the output.

INTRODUCTION

No software is perfect, and typically it does not work as anticipated when first used. In fact, programs running well for years may conceal undiscovered bugs. The probability of error escalates as software becomes more complex or is integrated with other software. Although there is no perfect software, the use of thorough testing procedures greatly reduces the number of possible errors. In addition, the earlier these testing procedures are introduced, the greater the developer's cost savings.

WHY CAN'T SOFTWARE BE PERFECT?

Software errors occur whenever software does not work as anticipated. Because some computer programs are many thousands of lines long, humans find it difficult to construct anything that large and complex without an error. Even errors that seem small can have disastrous consequences. For example:

1. A misplaced character on a single punched card led a New England town to believe that its tax base was $7 million more than it actually was. Because the tax rate was set too low, the town ran short of money.
2. Miscalculations fed into the Bureau of Reclamation computers kept too much water behind dams, resulting in floods along the Colorado River.
3. The Vancouver stock index lost one point per day for more than a year because of a computer error. By the time the error was caught, the index had lost 574 points.
4. A United Airlines jet lost power and altitude on a flight into Denver. This may have been caused by an onboard computer that did not adjust to dropping outside temperatures, causing the engines to ice up and then overheat.
5. The British destroyer *Sheffield* was sunk during the Falkland Islands War when the ship's defense system recognized an incoming Exocet missile as a friendly weapon and failed to shoot it down.
6. The maiden flight of the U.S. space shuttle was delayed because of an error in one of the 500,000 instruction lines.

All these examples had significant financial implications. William Goss, President of the International Bureau of Software Test, Inc. has observed computer systems running successfully for years before a bug appeared. For example, an equation failed after years of use because of one unique number set that had never been provoked before.

Alan Borning, a computer scientist from the University of Washington, estimates that a Star Wars defense system might require ten million lines of software code. He said, "You can't even test it adequately. You can't tell the Russians, 'Fire off a couple of missiles—we want to test our program.' It has to work right the first time without being tested in any real way."[1]

In *Software Testing Techniques,* Boris Beizer has said that a 10-character input string has 2^{80} possible input streams and corresponding outputs and noted that complete functional testing in this sense is clearly impractical. At one test per microsecond, it would take approximately four times the current estimated age of the universe![2] Thus, it is difficult for software to be perfect.

WHAT IS THE STATUS OF SOFTWARE TESTING?

Until recently, software testing has been hit-or-miss, with few standards and little consistency. Most professional testing occurred in a few large organizations in which special departments tested company-developed programs. Independent authors and small software development companies depended on in-house programmers or friends to test software. However, it is extremely difficult for software developers to test their own programs effectively because they do not make mistakes when entering data into their own programs. They have difficulty imagining the questions and confusion that novice users might experience. Friends of software developers also do not have the impartiality necessary to provoke and uncover bugs in their friends' programs. Thus, a number of new software programs do not receive thorough and objective testing.

Independent software testing laboratories can provide objective, fast, accurate, and economical testing because their primary intent is to identify software discrepancies as early as possible. Early intervention decreases testing and rework costs.

WHY ARE SOFTWARE TESTING PLANS AND PROCEDURES USED?

Software testing plans and procedures ensure that software discrepancies are reasonably provoked. They also ensure that all items are tested and verified and that tests can be repeated. Without testing procedures, test engineers must act on their insights at a particular moment. With testing plans and procedures, test engineers can repeat the exact sequence or keystrokes to reconstruct tests.

WHAT DOES A SOFTWARE TEST PLAN INCLUDE?

The test plan defines the project's scope and contains the test's objectives, goals, and methods. It outlines the product's features and includes the test schedule with time frame and milestones when the various testing stages are to be completed. The test plan also identifies the test levels, test types, and test methods.

WHAT ARE THE LEVELS OF TESTING?

The level of testing is the degree of testing to be used for each product module. The three primary levels are unit, system, and integration; for every project, one or more of these testing levels is defined in the testing objective.

Unit Level

The unit level of testing requires isolation of each unit from all other units and identification and control of interfaces with all other units.

System Level

All features and functions of only one system are tested in concert. The interaction between the system's units and the interfaces between the units are tested.

Integration Level

Integration level testing involves testing in concert all features and functions of more than one system. This could involve testing the system's intended configuration as a whole and include software, firmware, and/or peripherals. If several applications are possible, the integration level may involve testing all features and functions in relation to each other. For example, if a word processing, spreadsheet, spelling checker, and scheduling package are integrated, they all must be tested together.

WHAT ARE THE FOUR MAIN TYPES OF TESTING?

The four main types of testing include functional, destructive, regression, and documentation.

Functional Testing

Test suites designed for functional testing represent normal use by the intended user. Functional testing may involve checking limits, passwords, menus and screens, error messages, and prompts. It may test for clear messages and organization. It may also check for acceptable response time, proper interfacing, and whether all configurations work. In addition, it may check different combinations of functions. Finally, it may test the accuracy of error-handling routines and the use of exit routines.

Destructive Testing

Destructive testing includes abnormal use and conditions by the intended user. Areas for destructive testing include values outside boundaries, insufficient memory and/or storage, insufficient file structure, and data overflow. Finally, destructive testing checks software recovery from simulated hardware errors and rejection of invalid entries or combinations.

Regression Testing

Regression testing is a repeat of functional and/or destructive tests to determine that reported discrepancies have been corrected and that no new discrepancies appear as a result of updates or system corrections.

Documentation Testing

Documentation testing verifies that the documentation and the system match and that the documentation is well organized, easy to understand, and technically and grammatically accurate. Specifically, documentation testing checks whether the manual answers user questions and whether it communicates clearly and consistently.

WHAT ARE THE METHODS OF TESTING?

Software testing methods include procedural, outline, ad hoc, comparison, and compatibility testing.

Procedural Testing Method

Procedural testing uses strict guidelines for each test path so that every feature and function the documentation describes is tested. Test procedures are specific steps, keystroke by keystroke, that reach the result the test case defines. Test procedures provide reusable, consistent, step-by-step guidelines for future testing.

Procedural testing has the advantage of providing detailed documentation to test all product areas so that testers can recreate problem areas. The disadvantages of procedural testing are that it is time-consuming; it does not allow for intuitive testing; and if areas are overlooked in the testing design, testing may not provoke discrepancies in these areas.

Outline Testing Method

Testing engineers use outline testing to guide them through the testing of interrelated parts of a system. All features listed in the outline are tested, but not necessarily in the order of appearance. Outline testing is partly an intuitive testing method, because the tester follows logical paths wherever they lead. Testers prepare test logs that provide a reference to the flow and logic of testing.

Ad Hoc Testing Method

Highly skilled testing engineers use ad hoc testing for specific areas. No documentation, other than software discrepancy reports, is produced; no guides are followed other than the testing engineer's intuition and the supplied documentation.

Comparison Testing Method

Comparison testing determines the similarities and differences between two or more systems. It may involve taking the reference section from the accompanying documentation, building a feature comparison matrix, and then testing each command on one system and then on the other system. This is followed by comparing the results and documenting any differences.

An alternative, if the systems under test are compatible, is to run an automated test simultaneously on each system, comparing the results and documenting any differences. If external hardware such as printers is involved, comparison testing can be accomplished by creating a master document that uses all possible features and functions and producing output on each printer. Then results are compared and differences are documented.

Compatibility Testing Method

Compatibility testing determines the ability of two or more systems to function interchangeably and identifies any functional differences. Areas of focus include keyboard input, CRT display, reproducible discrepancies, operating system functions, I/O functions, and disk exchange.

WHAT ARE SOFTWARE STANDARDS?

Software standards may be as formal as an established industry standard such as the "ANSI FORTRAN 77 Standard" or the information specifications to which the product was designed. The International Bureau of Software Test supports the ACM and the IEEE in establishing and defining software testing standards.

WHAT ARE TESTING PROCEDURES?

Testing procedures are detailed documents describing specific steps for accomplishing each test's goals. They include criteria for acceptance or rejection, as well as the configuration required to perform the test. Although the procedure's complexity varies with the testing level, the form of the procedures is similar. Software testing procedures require the test staff to review the entire software product for content, presentation, organization, and adherence to documentation. These procedures require manual inputs, visual review, or a combination of the two. Execution testing can consist of manual input, semiautomatic input, and/or automatic input.

WHAT ARE ADVANTAGES AND DISADVANTAGES OF VARIOUS TESTING PROCEDURES?

Manual Input

Test engineers manually enter data and control information as described in the test procedures. The primary advantage of manual input is the direct flow from the procedure's development to the test engineer for test execution. The main disadvantages are that (1) manual input of data introduces the possibility of an input error and (2) there must be continuous staffing for the total period of test execution and evaluation. This staffing requirement affects the testing cost because every time the tests are run, the investment in staff to run the tests remains the same.

Semiautomatic Input

Semiautomatic input consists of part manual procedures and part software test code execution. Primary advantages of semiautomatic test procedures are (1) the limited need for developing test code and (2) the reduced requirement for staff attendance and intervention. These advantages provide cost control. The main disadvantage of semiautomatic execution is that manual input of data introduces the possibility of an input error.

Automatic Input

The test engineer executes the test code, including the automatic test result verification, and reviews the results. The major advantages of automatic input are (1) the ability to repeat the test in exactly the same way every time, (2) the greatly reduced requirement for staffing during test execution so that the tester can start the test and review the results when the test is completed, and (3) lack of human error. The primary disadvantage of automatic input is the cost of investment in test code and test tool development before the test is executed. However, it must be recognized that this high cost is amortized over the number of times the test is performed.

WHEN SHOULD TESTING PROCEDURES BE INTRODUCED?

The single largest component of software cost is testing to discover software bugs. In fact, testing costs can exceed 50% of the software development expense. Therefore, software testing procedures should be introduced as early as possible in the development cycle. The earlier errors are found, the less expensive they are to correct. If testers review design specifications, they can call attention to problem areas and look for shortcomings. Testers can also be developing test plans while programmers are programming so that when coding is completed, testing can begin.

CONCLUSIONS

Although there is no perfect software, the use of thorough testing procedures greatly reduces the number of possible errors. In addition, the earlier testing procedures are introduced, the greater the resulting cost saving.

REFERENCES

1. Borning, A. "'The Last Bug' Computer Scientists Fear." *San Jose Mercury,* October 29, 1984, p. 1.
2. Beizer, B. *Software Testing Techniques.* New York: Van Nostrand Reinhold, 1983.

A view on windows: Current approaches and neglected opportunities

by BENN R. KONSYNSKI, ARNOLD GREENFIELD, and WILLIAM E. BRACKER, JR.
University of Arizona
Tucson, Arizona

ABSTRACT

This paper offers a taxonomy of current approaches to the use of windows in screen dialogues of application systems. Windows are viewed as a mechanism for managing presentation space in the design of information systems. Several neglected application areas are explored, including the use of mulitple windows in the design of single applications. The potential and limitations of using windows is also discussed.

INTRODUCTION

As the office becomes computerized, we are seeing increased use of sophisticated software packages in all areas of the business environment. For example, word processors help secretaries prepare letters, while spreadsheet programs help both accountants and strategic planners. However, even with all their power, many of these packages are failures because they are difficult to use and present their results poorly. As a result, research has proceeded in earnest towards improving both the elicitation and presentation of information between man and computer.

Information systems design has generally emphasized *function* while neglecting *form,* yet designing the format of the interface is just as important to system success, and may prove even more difficult than identifying and specifying functionality. Specifically, designers have had to work with a limited presentation space constituted by the viewport onto the system. That presentation space is the loci of the system's presentation of information to the user, and the user's clarification of communication with the system. As the central focus of the dialogue, it has often determined the limits of the user/system dialogue. Neglecting the management of this space results in system resistance, and encourages situations where dialogue is a constraint on effective use, rather than a facilitator.

Many breakthroughs in user interface design have resulted from studying the physical work environment, and determining whether management can make effective use of the growing computing capability. This has resulted in the adoption of certain motifs that form the basis for dialogue design in office systems. For example, when working at his/her desk, a manager generally has several items present; a pad of paper may be in front of him; a book may be open on his left; and a series of graphs indicating the previous year's sales performance may be on his right. Such an arrangement can be translated to the computer in that the screen can be divided up into several areas or windows, with the user being able to work on different tasks in each of the windows rather than being limited to one task interaction at a time (as is currently the trend). As the computer environment more closely resembles the manager's work environment, it should be easier for the manager to use the computer in a decision support capacity.

The notion of splitting the screen into multiple windows goes well beyond the desk paradigm. Windows help users to organize their work and integrate several subtasks. Current interfaces are generally set up for performing these subtasks sequentially. However, we know that users often work in parallel, switching their attention between several items. By allowing the user to interact with various subtasks on the same screen, windows help the user to integrate the results of the subtasks and thereby complete the overall task.

Office workers are frequently involved in the monitoring of change, or find themselves waiting for important messages. Managers are often interrupted or alerted to important events in the organization. The manager's connection with the electronic network in the organization is a limited port or presentation space for such transactions, but windows and icons provide the manager with multiple viewports for situations that are normally handled by phone calls, distracting physical interruptions, untimely notes, and other forms of interruption. Windows can allow the user to accept, prioritize, or delay such interruptions, alerts, or reminders.

As might be expected, windows also have certain drawbacks. Many users have complained that windows can quickly become too small to adequately display information. Some complain about the slowness of the system. Others question the usefulness of working with more than one task at a time, citing limitations in short term memory.[1] Market performance of currently available window packages seems to echo this sentiment; sales are low and vendors are drastically reducing their prices (for example, one vendor package was reduced from $495 to $95).

As we shall see, the reasons given for user disappointment are only partially founded. In particular, the most likely reason for the dismal performance of window packages to date is not the infeasibility of windows per se, but how they have been applied and implemented. This paper reviews the windowing concept, details the current environment and discusses how appropriate window management can improve system utility and usability. We first give a brief history of windows, then review several of the current window support environments, discuss certain neglected opportunities for window use, and finally describe the benefits and limitations of windows.

A BRIEF HISTORY OF WINDOWS

One of the first applications to use windows was a text editor developed by Engelbart in 1973.[2] The overall task context was well understood; the windows were useful in helping the user concentrate on the work at hand while keeping the overall problem context in the forefront. In 1976, windowing was extended to a multi-tasking environment by Buneman and colleagues in the implementation of the DAISY system.[3] Several independent tasks could be displayed on one screen, and the user could direct input to and receive output from any one of them. However, transferring data between windows, an important part of task integration, was not well defined.

The first formal window environment available commercially was the Xerox STAR.[4,5] The STAR displayed a desktop composed of various icons, such as an in-box, out-box, and file cabinet. When "opened," the icons became windows on the screen, through which the user could interact with the system. Several of the techniques for manipulating windows were quite innovative, and were carried over into future window packages. Unfortunately, the STAR initially proved to be rather expensive for general users.

The real breakthrough for windowing occurred when windows became feasible for microcomputers. Due to the improvements in microprocessors and reduction in memory costs, software vendors rushed to develop window packages that would "revolutionize" the design of user interfaces. It is the availability of these new systems that has sparked both a renewed interest in windowing opportunities, and a glut of disappointing implementations.

We next discuss several of the initial directions that software vendors have taken in their attempts to provide window support for microprocessor-based applications and office environments.

A REVIEW OF CURRENT WINDOW ENVIRONMENTS

The windowing concept has been exploited by designers in various ways. Alternative designs reflect the various skills or technologies that were brought into play when developing the windowing capability. Three basic approaches for window use have surfaced.

1. Integration within the operating system
2. Multiple-application support
3. Business shell environments

Let us briefly examine each of these approaches.

Windows Central to the Operating System

Making windows a central mechanism to interface with the operating system is becoming increasingly popular. For example, Xerox, Apple, and Apollo have window-oriented operating systems. These systems make use of windows, icons, and often a mouse to facilitate the presentation, elicitation and identification of operating system command functions. These systems require the presence of high resolution displays to exploit window and icon concepts.[6]

With the STAR, users can open one or more icons, resulting in several partially overlapping windows on the same screen. The STAR design goal was to make everything relevant to the current task visible on the screen;[7] therefore, the desktop, while frequently obscured, always remains on the screen. This design goal is instrumental in helping the user organize his subtask while keeping the overall task in mind.

The Lisa and Macintosh from Apple Computer have improved upon some of the STAR concepts to aid task integration. In addition to using icons to indicate system functions, Lisa and Macintosh allow users to transfer information between windows. For example, the user can work on the text for a document in one window, and draw a figure for the document in another. The figure can then be transferred to the text window, appropriately positioned in the document. The mechanisms to indicate such transfers are more easily identified in the Apple implementation than were accommodated in earlier implementations.

Lisa and Macintosh (and several other systems) also make use of the pull-down menu, where lists of options appear in a window that partially obscures current windows. Once the user has made a choice, the menu disappears, again revealing the overlayed windows. This technique helps the user visualize current options in the context of the overall task.

The Apollo system's windowing is somewhat different and less advantageous than that of Xerox or Apple in that its various windows do not have a unifying concept such as a desktop. Rather, each user-created window functions as a separate terminal. As a result, task integration is not as strong with Apollo as with Xerox and Apple. Apollo also uses windows for specific purposes. For example, one window is used to input system commands; another is used only for system error message display. Further, the Apollo system has not taken advantage of alternate input devices as Apple has with its mouse. With early versions of the Apollo system, many functions required keystrokes that could have been translated into motion, were other input devices used. We expect that this problem is being addressed in current installations.

In summary, different tasks may run in different windows, or as with the Apollo system, windows can serve to logically divide inputs and outputs. Such techniques are aimed at helping the decision-maker to better communicate with the computer and thereby organize his work; initial results to this point have been mixed.

The integration of windowing capability for dialogue interface in the operating system may prove to be a major boon that increases the general usability of various systems. For users of current operating systems, the functional activities are easily understood, yet the syntax and semantics of the interface terminology serve to inhibit and disinterest them. Building window functioning capability into sharable code in ROM for support of basic operating system interface requirements and application management support may, in the long run, surface as one of the most valuable contributions to making computers more accessable to the public.

Multiple-application Window Packages

While window packages that support multiple applications in different windows are often similar to their operating system counterparts, they have two important differences:

1. One of their main goals is to help integrate diverse software packages and provide for data-flow between packages.
2. Most such packages are built to run as shells on top of existing operating systems.

Two of the more popular multiple-application window packages are DesQ by Quarterdeck, and Windows by Microsoft. IBM recently announced Topview, a window package designed for the IBM PC.

DesQ and Windows are very similar in that they allow diverse software applications to be run in different windows. To aid task integration, DesQ has elaborate protocols for transferring information between windows. Quarterdeck depends on "local experts" at corporations where DesQ is installed to set up these transfer protocols, a strategy that may discourage quick acceptance of the product.

DesQ is claimed to be able to run a majority of IBM-compatible programs; Windows requires existing programs to be rewritten before they will run. Specifications for the recently released Topview are vague, and will remain so until the product is in distribution.

Many of these and related packages require and exploit RAM-disk capabilities to realize the redirection of input and output, and to facilitate the buffering required for viewport data management. Without such facilities, implementation would be intolerably slow (e.g., piping in MS-DOS). Still, few such implementations have met with much market success.

Business Shell Environments

Within the set of business shell environments, vendors integrate well-defined packages rather than diverse applications. Reflecting a user trade-off of flexibility for the power of integration, the applications that run in different windows of these shells are well defined; as such, the transfer protocols between windows are easier to understand and use. However, the user is limited to the small set of applications that come with the business shell. Frequently, one or more of such applications are not as good as similar products on the market.

Three of the more popular business shell environments are Framework by Ashton-Tate, Symphony by Lotus Development Corporation, and VisiOn by VisiCorp. These shells provide similar capabilities, such as spreadsheets, word processing, database management systems, graphics, and communication software. Framework also provides an outline function (which makes Framework arguably superior to the other two packages) that allows users to create documents with outlines; entries in the outline, such as a graph or a spreadsheet item, can be from different component packages. The user can alter the document by changing the outline; the underlying parts of the outline are automatically moved.

All three of these business shells are useful, but it is not clear that the user needs all of the power provided by them. User work area size is often a problem, because most packages store all applications in main memory. The idea of close task integration is good, but lessening the number of packages while improving their quality might be a better option. A major problem associated with this approach is that each package has grown around the success of one subportion of the task activities (e.g., spreadsheet or word processing), and the others are only half-heartedly supported. This imbalance may be addressed in subsequent releases of these packages.

The OfficeWare system, available through NCR Corporation, offers a somewhat different approach to business shell design by redistributing the functions that are associated with dialogue and functional-processing between the workstation, or PC, and a host server. For example, in the MIS Department at the University of Arizona, an OfficeWare installation runs with NCR PC4s and IBM PCs as workstations, and an NCR Tower (UNIX) system as a server. The screen management functions that are best handled at the PC level are allocated to the PC environment, while the basic business functional applications are associated with the server function.

In examining the above environments, we notice an interesting dichotomy. Those environments that promise "mainframe-like" windowing capabilities (such as VisiOn) have generally not fared well, probably because the microcomputer is still not powerful enough to satisfactorily mimic the more expensive devices. However, those systems that understand the limitations of the current environment, and concentrate on exploiting the capabilities within these limitations have fared much better (e.g., Macintosh). An obvious comment on the MacIntosh is that the reasons it proves successful in the delivery of effective windowing precluded a degree of access to existing business software growing in the MS/PC-DOS environment.

We further note that the focus of current environments is on the use of windows to support multiple applications, either for operating system functions, or presentation of simultaneous displays. We believe that a neglected opportunity is the support for multiple windows through a single application.

SUPPORT FOR SINGLE-APPLICATION, MULTIPLE WINDOWING

It is the intent of most window support environments to provide for windowing across applications. In micro-environments, this usually takes the form of window support in RAM-disk packages that allow each application to maintain a single open window, and provides for piping between applications. While this form seems to be desirable for the integration of applications, it offers many problems with current and near-term technology environments.

Designers, who often fail to appreciate current microcomputer limitations, have concentrated on highly sophisticated window packages while overlooking the opportunity to use multiple windows in a single application setting. Such technological considerations are not the only reasons for advocating the single-application/multiple-windowing approach. With the advent of dialogue protocols that involve currency indicators, switches, clocking, status reporting, and other message transfers between the system and the user, there is a clear need for multiple windowing in order to make the dialogues more effective in terms of the use of a limited presentation space. The result is a system that is more maintainable, flexible, standardized in dialogue protocols (e.g., all system messages on bottom line, etc.), and easier to debug.

If we look back for a moment, we can analyze how the present state of affairs came into being. The traditional form of windowing involved the use of a single viewport for a single application. However, in some environments, one was allowed to have a single viewport with multiple applications operative, as in multi-tasking environments where only one active process can be viewed at a time. From this situation, designers recognized the opportunity to exploit the viewing space for multiple simultaneous observations. However, the mistaken perception in multi-application multiple-windowing is one of numbers; a typical user will have no more than two or three applications active at any time. Screens get so cluttered that one loses track of ongoing activities when too many viewports are *open* or *visible.*

The key issue involves the recognition by developers of the relationship between the functional application needs of the user (i.e., a single- or multiple-application task active at any given time), and the technology, which is focused on a single- or multiple-windowing capability.

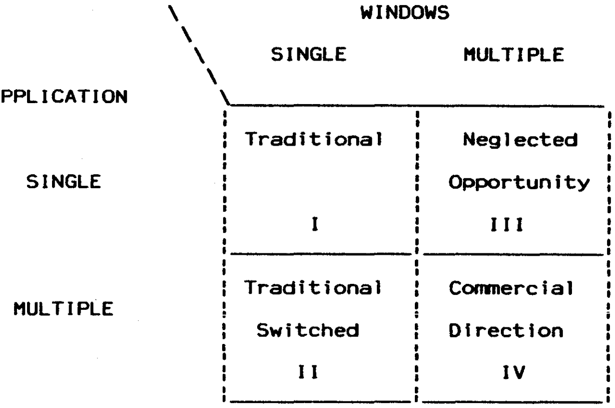
Figure 1 illustrates the various directions taken in the issue of multiple windows and applications. The neglected opportunity (III) in this classification is the use of the multiple-windowing capability to support the multiple information presentation needs in single applications. In fact, most effective interactive systems already use built-in screen management utilities to present multiple pieces of information during the execution of an application. Wordstar is one such example, with small status and currency windows displayed on the head of the screen followed by menu information which is in turn followed by the editing space that viewports on the editing buffer. In effect, there are several manageable windows that are displayed simultaneously and updated as if multi-tasked.

The use of multiple windows in a single application allows for better and more flexible design of the interface (e.g., windows can be moved independent of other application functions); better separation of presented information is available, reducing the cognitive demand on the user; further, the use of multiple windows offers certain clarity in screen management that may significantly simplify the design process and facilitate earlier implementation.

EXAMPLE USE OF MULTIPLE-WINDOWING IN MASH SYSTEM

An example of the use of window support in a single application environment is the electronic mail support facility called *MASH* (for Mail Access Supporting Heterogeneity).[8,9] MASH was developed to allow users on PCs (IBM, NCR, etc.) to interact with multiple mail servers (VAX, Tower, ARPA, etc.) over multiple communications facilities (SYTEK, MODUS, voice-grade lines, etc.). Thus, with a single dialogue protocol, a user of a micro-based station is able to connect to multiple mail systems without having to deal with strange and volatile protocols.

MASH uses windows for screen and display control. Multiple windows in monochrome or color are used to control the user/system dialogue. For example, Figures 2 to 4 illustrate system interaction with the user. Figure 2 shows the initial screen at startup with a user profile window also displayed. In Figure 3, the user (John Smith) has created a mail message to be sent to Jones. Windows indicate header information, text of the message, and commands available at the present time. Figure 4 shows entries in the user's "in-box." Note that the commands have changed, and the header information has been replaced by the in-box directory.



APPLICATION \ WINDOWS	SINGLE	MULTIPLE
SINGLE	Traditional I	Neglected Opportunity III
MULTIPLE	Traditional Switched II	Commercial Direction IV

Figure 1—Overview of directions taken (and not taken) in multiple window applications

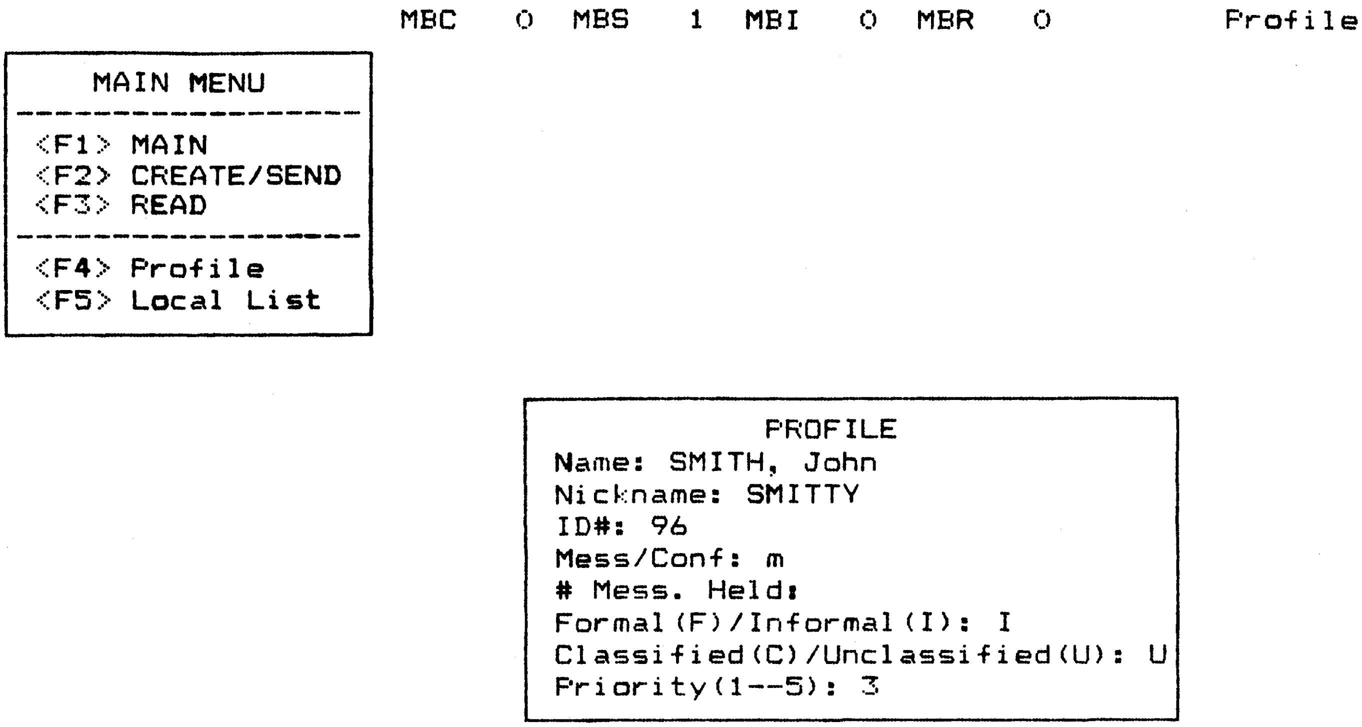
The window management system that underlies the MASH package controls window presentation and placement. The user can change the location or presentation of windows without modifying the functional software of the application. Indeed, a callable window editor and field editor are used to standardize the interface between windows. Windows allow the user to focus on a windowed class of information in a read, as well as read/write organization. Scrolling and other non-destructive editing functions are also available in read-only windows.

MASH is an example of a package that takes advantage of the windowing function in the design of single applications. The application is more easily understood and quickly learned by the user. Modification and adjustment of the system are also facilitated through the use of the window management support that offers an independence for the dialogue similar to that offered for data management through the use of database management systems. The same window and callable editor functionality is available on a wide variety of micros and facilitates a significant degree of interoperability.

Thus with MASH, we demonstrate the ability to establish systems that provide effective window support across processor and operating system environments. Furthermore, the user can have a window picked up, resized, and placed elsewhere without affecting the application logic. In the absence of multi-tasking, as in the MS/PC-DOS environment, we are provided with a perception of multiple-task management through the currency indicators and alerting functions.

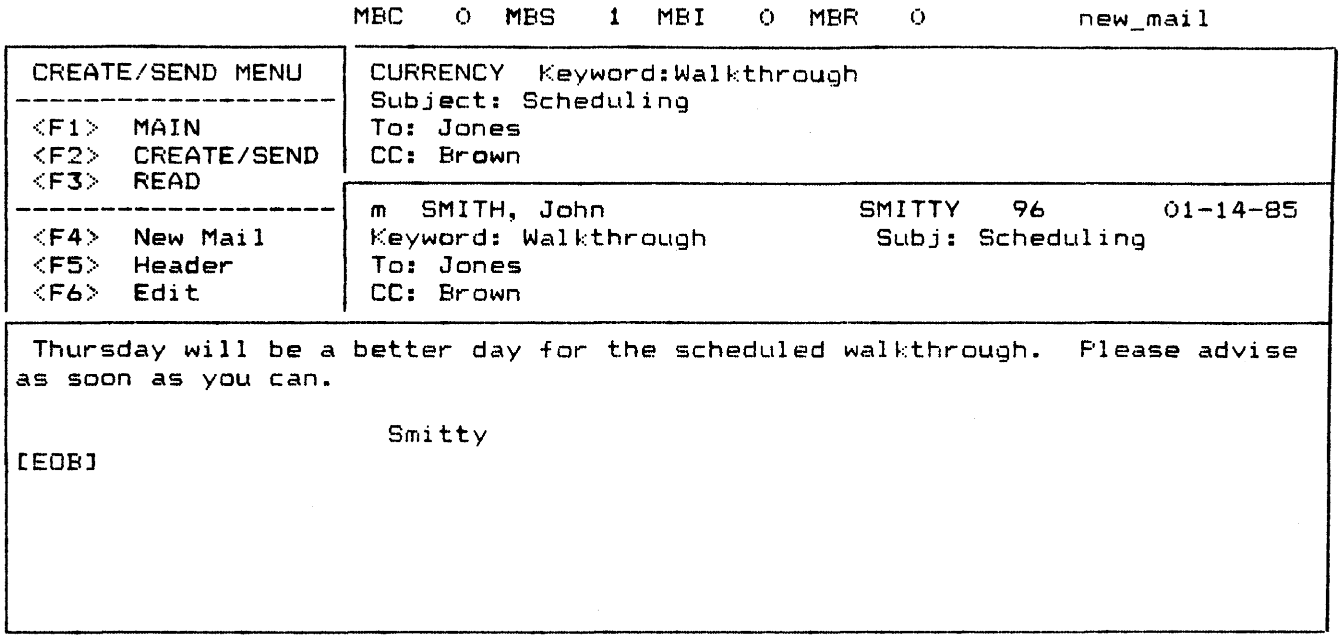
We have made other use of windowing in the current implementation of the PLEXSYS system (see Figure 5),[10] and for an inference engine in an application of artificial intelligence in a clinical pharmacy.[11] In these cases, we have found that windows were useful for

1. organizing the presentation and elicitation of information;
2. offering a flexibility in coding; and
3. providing a framework for the architecture of the software system.



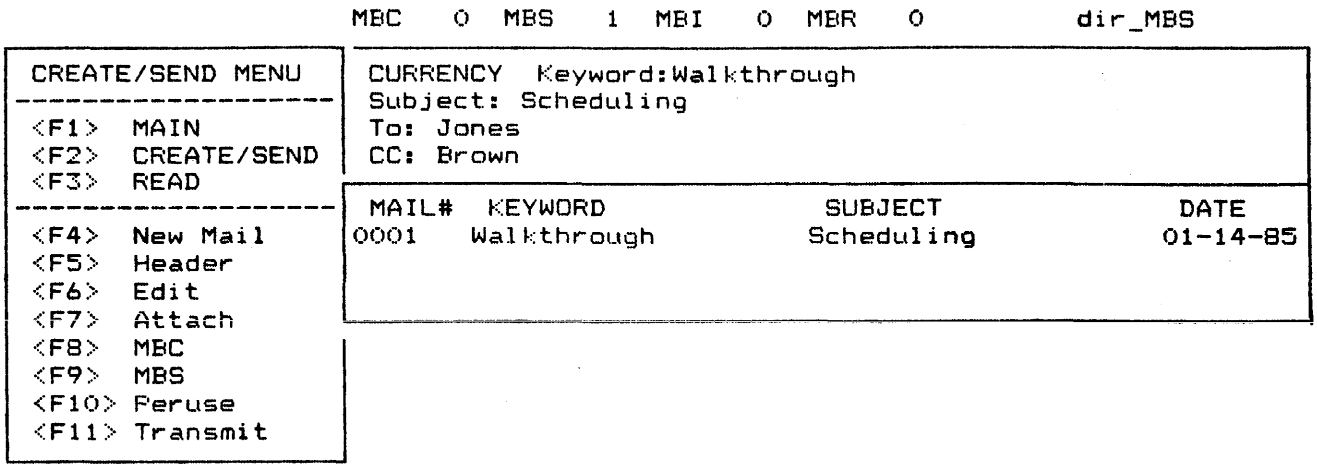
```
                     MBC    0  MBS    1  MBI    0  MBR    0             Profile
MAIN MENU
-------------------
<F1> MAIN
<F2> CREATE/SEND
<F3> READ
-------------------
<F4> Profile
<F5> Local List

                          PROFILE
                   Name: SMITH, John
                   Nickname: SMITTY
                   ID#: 96
                   Mess/Conf: m
                   # Mess. Held:
                   Formal(F)/Informal(I): I
                   Classified(C)/Unclassified(U): U
                   Priority(1--5): 3
```

Figure 2—MASH entry screen with user profile window visible



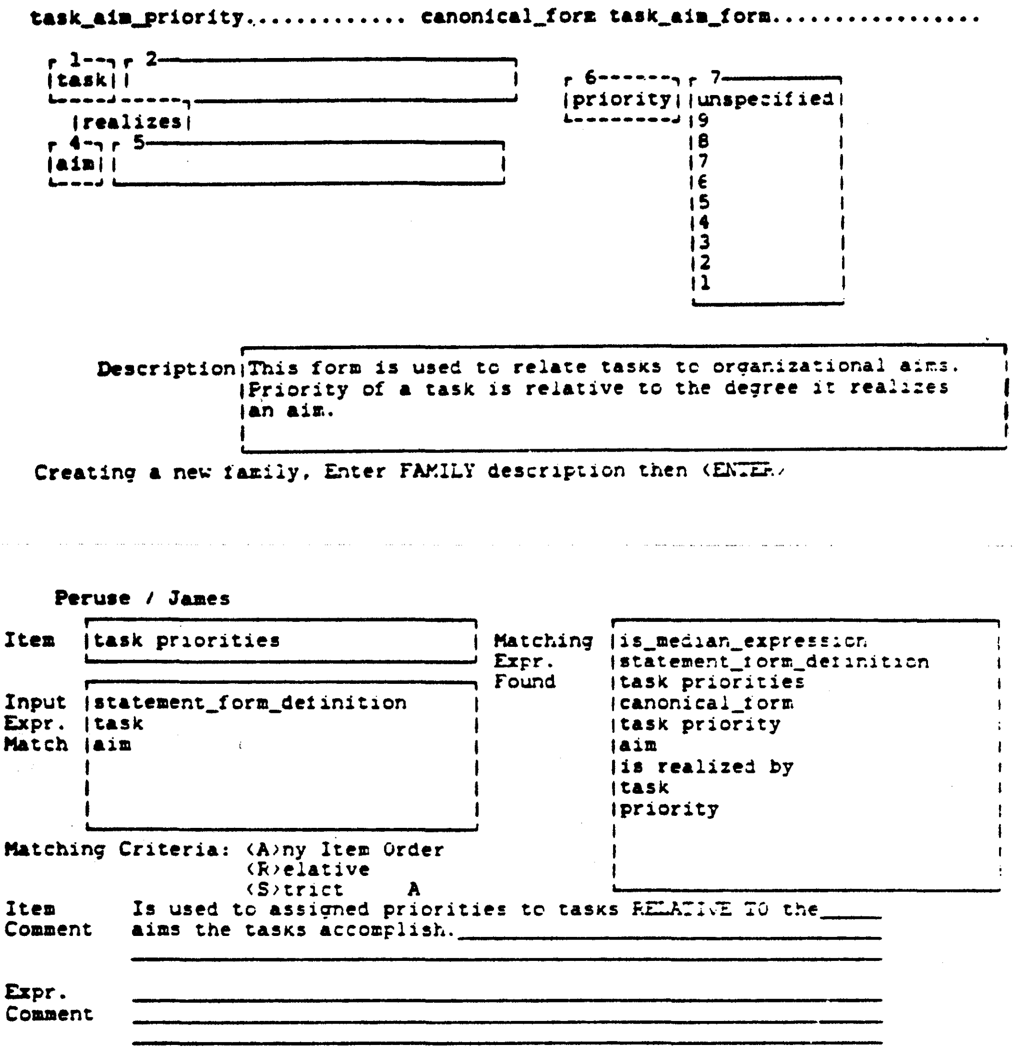
```
                   MBC    0  MBS    1  MBI    0  MBR    0          new_mail
CREATE/SEND MENU   | CURRENCY  Keyword:Walkthrough
------------------ | Subject: Scheduling
<F1>  MAIN         | To: Jones
<F2>  CREATE/SEND  | CC: Brown
<F3>  READ         |
------------------ | m  SMITH, John                SMITTY   96      01-14-85
<F4>  New Mail     | Keyword: Walkthrough           Subj: Scheduling
<F5>  Header       | To: Jones
<F6>  Edit         | CC: Brown

 Thursday will be a better day for the scheduled walkthrough.  Please advise
as soon as you can.

                     Smitty
[EOB]
```

Figure 3—MASH display with header, currency and message windows



```
                              MBC   0  MBS   1  MBI   0  MBR   0          dir_MBS

CREATE/SEND MENU     CURRENCY  Keyword:Walkthrough
-------------------  Subject: Scheduling
<F1>  MAIN           To: Jones
<F2>  CREATE/SEND    CC: Brown
<F3>  READ
-------------------  MAIL#  KEYWORD              SUBJECT               DATE
<F4>  New Mail       0001   Walkthrough          Scheduling           01-14-85
<F5>  Header
<F6>  Edit
<F7>  Attach
<F8>  MBC
<F9>  MBS
<F10> Peruse
<F11> Transmit
```

Figure 4—MASH display with a mailbox window visible



```
task_aim_priority............ canonical_form task_aim_form................

┌ 1--┐┌ 2──────────────────────────────┐
|task||                                |     ┌ 6------┐┌ 7──────────┐
└----┘-----┐──────────────────────────┘     |priority||unspecified|
   |realizes|                                └--------┘|9          |
┌ 4-┐┌ 5────────────────────────────┐                   |8          |
|aim||                              |                   |7          |
└---┘└──────────────────────────────┘                   |6          |
                                                        |5          |
                                                        |4          |
                                                        |3          |
                                                        |2          |
                                                        |1          |
                                                        └───────────┘

           ┌──────────────────────────────────────────────────────┐
Description|This form is used to relate tasks to organizational aims. |
           |Priority of a task is relative to the degree it realizes |
           |an aim.                                                  |
           |                                                         |
           └──────────────────────────────────────────────────────┘
Creating a new family, Enter FAMILY description then <ENTER>
```

```
    Peruse / James
        ┌──────────────────────────┐           ┌──────────────────────────┐
Item    |task priorities           | Matching  |is_median_expression      |
        └──────────────────────────┘ Expr.     |statement_form_definition |
        ┌──────────────────────────┐ Found     |task priorities           |
Input   |statement_form_definition |           |canonical_form            |
Expr.   |task                      |           |task priority             |
Match   |aim                       |           |aim                       |
        |                          |           |is realized by            |
        |                          |           |task                      |
        |                          |           |priority                  |
        └──────────────────────────┘           |                          |
Matching Criteria: <A>ny Item Order            |                          |
                   <R>elative                  |                          |
                   <S>trict     A              └──────────────────────────┘
Item      Is used to assigned priorities to tasks RELATIVE TO the_____
Comment   aims the tasks accomplish.__________________________________
          ____________________________________________________________

Expr.     ____________________________________________________________
Comment   ____________________________________________________________
          ____________________________________________________________
```

Figure 5—Sample screens from PLEXSYS system illustrating multiple windows

The third point suggests the specialization of language constructs for windowing as a means of guiding much of the structure of the logic in system specification.

THE POTENTIAL AND LIMITATIONS OF WINDOWING

As the technology continues to improve and becomes more readily available, windowing will assume a broader role in the structure of user/system dialogues. We can expect to realize more of this potential, but to what limits? This section reviews issues related to the potential and limitations of windowing.

Windows offer a useful capability that certainly addresses many of the complaints that users have had regarding the nature of their user/system dialogues. At the same time, windows are merely one tool at the disposal of dialogue designers,[12, 13] not the answer to all concerns.

Some benefits of windows are well known; they indicate the presence of multiple-activity tasks; they reflect the preservation of activities, even when sequential-tasking takes place; furthermore, they serve to separate items that should be separated. It is no minor benefit that this separation offers a more aesthetically pleasing interface.

There are many other benefits to windowing, including the opportunity to gain multiple views of the same data in different perspectives (i.e., a graphic and tabular presentation). In addition, the "tidiness factor" encourages cognitive tidiness and screen management "cleanliness." Piping from one window to another allows the direct perception of multiple-task management, and addresses the often-cited user concern for application interconnection. Some other benefits of windowing include, but are not limited to the following.

1. Improved perception of multi-tasking;
2. Separation in space and identity (border, color, etc.), thereby easing cognitive demand;
3. Ability to make direct comparisons;
4. Displays of currency indicators;
5. Displays of alerting or exception information;
6. Isolation and separation of functional displays (commands, messages, etc.); and
7. Framework for organizing dialogues and procedures specifications.

The limitations to current windowing approaches are partly due to increased user expectations and the over-sell of the ability to mimic expensive interfaces (e.g., the case of the Xerox Star) on inexpensive devices (e.g., slow processors like the 8088; limited display capabilities such as terminal I/O; and keyboard input). Thus, the market is over-sold on the capability and disappointed in its delivery. This dichotomy could have been avoided had proper care been taken in the introduction of the concept and capability.

Another major problem in the use of windowing is the nature of the display and I/O. Where bit-mapped graphics and direct memory addressing is available, the windowing operation is fast and efficient, but attempts to use windows in environments where BIOS operations and other software layers are used creates excessive overhead. Thus, in a majority of MS-DOS environments where such layers are involved, and the screen management is character-oriented, system performance as a multi-application, multi-windowing tool is dismal.

Other limitations of currently installed windowing functions are as follows.

1. Interfaces are designed for lowest-common-input mechanisms, usually limited to the keyboard. Use of the mouse is just emerging in inexpensive environments;
2. Windowing is usually only offered in RAM-disk environments;
3. Window packages frequently use standard I/O facilities with significant overhead;
4. Current packages do not provide the hooks for single-application use of multiple windows;
5. Screen resolution is often limited, reducing potential window functionality;
6. Limited character graphics are often used for windowing support; and
7. Little thought has been given towards helping design window interfaces in specific application domains.

These and other problems need to be resolved before windowing benefits will accrue to the user community. We have seen the promise in the early workstation implementations, yet the inability to translate beneficial features in these environments to the small, inexpensive desk-top environment has resulted in a pervasive disappointment with the ability of windowing to make a difference in the functional utility of systems in the marketplace. The skeptical attitude on the part of potential users should lead developers to recognize that benefits of a windowing environment can be realized when certain minimal standards of functionality in the workstation environment are met.

CONCLUSION

Windowing offers significant potential for information organization and presentation control. Its functional capabilities suggest the means of integrating and defining applications in the context of window dialogues. We can expect that interest in windows will grow as support technologies become less expensive, and that window support will be a commonly exploited technology in operating systems, applications, and utilities.

This paper has reviewed several of the current windowing directions and briefly examined efforts in the commercial arena. We have also tried to bring to light the neglected opportunity presented by multi-windowing in a single application environment. Clearly, the potential benefits of windowing in user/system dialogue environments are many. The race by application designers to take advantage of emerging technology, and the costs of technology delivery will result in efficient, effective windowing environments in future information systems.

Windowing is one of many seldom-used technologies that dialogue designers can bring to play to promote effective dialogues that enhance the utility of information systems. Its promise is not that of a guarantor of more useful and effective systems, but surely that of a major player in the process of making information systems more useful and effective.

REFERENCES

1. Miller, G. "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information." *Psychological Review,* 1956.
2. Engelbart, D. "Design Considerations for Knowledge Workshop Terminals." *AFIPS, Proceedings of the National Computer Conference* (Vol. 42), 1973, pp. 221–227.
3. Buneman, O., H. Morgan, and M. Zisman. "Display Facilities for Decision Support: The DAISY Approach." *Database,* 8-3, (1977).
4. Bewley, W., T. Roberts, D. Schroit, and W. Verplank. "Human Factors Testing in the Design of Xerox's 8010 'Star' Office Workstation." *Proceedings of CHI Conference,* ACM, 1983, pp. 72–77.
5. Lipkie, D., S. Evans, J. Newlin, and R. Weissman. "Star Graphics: An Object-oriented Implementation." *Computer Graphics,* 16-3 (1982), pp. 115–124.
6. Goldberg, A., and D. Robson. *SMALLTALK-80: The Language and Its Implementation.* Reading, Mass.: Addison-Wesley, 1983.
7. Smith, D., C. Irby, R. Kimball, B. Verplank, and E. Harslem. "Designing the Star User Interface." *BYTE,* April 1982.
8. Management Information Systems Department, University of Arizona. *MASH Users Manual.* 1984.
9. Chen, R. B., B. Konsynski, and F. Y. Kuo. *A Callable Library of Window Management Utilities.* Management Information Systems Department Technical Report, University of Arizona, Tucson, June 1984.
10. Konsynski, B., J. Kottemann, J. Nunamaker, and J. Stott. "PLEXSYS-84: An Integrated Development Environment for Information Systems." *Journal of Management Information Systems,* Spring 1985.
11. Konsynski, B., and D. Heer. *An Application of Artificial Intelligence in Clinical Pharmacy.* Management Information Systems Department Technical Report, University of Arizona, Tucson, 1983.
12. Kuo, F. Y., and B. Konsynski. "Dialogue Management in Information Systems." TIMS/ORSA Meeting, 1983.
13. Kuo, F. Y., and B. Konsynski. *Dialogue Management Support Environments in Information Systems.* Management Information Systems Department Technical Report, University of Arizona, Tucson, 1985.

Artificial intelligence on personal computers*

by DAVID R. BRODWIN**
Arthur D. Little, Inc.
San Francisco, California
WAYNE ERICKSON
Microrim Inc.
Bellevue, Washington
JERROLD KAPLAN
Teknowledge, Inc.
Palo Alto, California
and WANDA RAPPAPORT
General Research Corporation
McLean, Virginia

ABSTRACT

In the past year, the capabilities of artificial intelligence have begun to appear in personal computer software products. This transfer of technology holds the potential to broaden significantly the market for AI and radically transform the PC marketplace. This paper presents a framework for understanding and evaluating current PC-based AI software; discusses the hardware, software, and conceptual roadblocks to further developments; and projects future trends in the converging arenas of AI and personal computing.

* This paper discusses general developments in the industry. The suggestion herein that particular products or capabilities may be feasible or are under development implies nothing whatsoever about the product plans of the companies with which the authors are affiliated.

** While the last three authors contributed substantially to this paper, the lead author drafted and edited the final document. Accordingly, he takes full responsibility for any accidental errors or omissions and for all viewpoints not otherwise attributed.

INTRODUCTION

In few areas is technology expanding our horizons as dramatically as in the converging fields of artificial intelligence and personal computing. The past year has seen the introduction of more than a dozen personal computer products that incorporate or support artificial intelligence concepts and techniques. These products range from programming languages to expert systems development tools to end-user software products. Considerable press attention has followed these developments, both attacking them as hype and praising them as the start of a new era of personal computing.

This paper discusses the transfer of AI technology to personal computers. It provides a framework for understanding the range of PC AI software available today (and separating the true AI software from the masqueraders), identifies the major driving forces in the industry, and provides some projections about the future.

This paper is oriented towards the relatively non–technical reader; accordingly it deals more with broad issues and trends than with technical descriptions of the products discussed. Our methodology consisted of interviews with leading participants in the PC AI industry; the data and perspectives thus obtained were analyzed by using criteria of technical feasibility, user requirements, and the structure and dynamics of the personal computer software market as it exists today.

TYPOLOGY OF ARTIFICIAL INTELLIGENCE PRODUCTS FOR PERSONAL COMPUTERS

The range of announced and rumored PC AI products can best be understood if viewed as a hierarchy of languages, tools, and applications (See Figure 1). This framework allows one to classify products and evaluate them realistically against others of their category. Such classification and evaluation is unfortunately necessary, because a number of conventional software products have started masquerading as artificial intelligence in order to capture the publicity AI now enjoys. For the purposes of this paper we define artificial intelligence in software as the use of any of the techniques developed by, and associated with artificial intelligence researchers, (e.g. symbolic reasoning, fuzzy sets, or natural language processing through augmented transition networks).

Language Products

In the past year, the standard artificial intelligence languages, notably Lisp and PROLOG, have been ported to the personal computer environment. Several versions of each are now released or under development; for example, the Golden Common Lisp, developed by Golden Hill computers, and the Macintosh Lisp under development by Expertelligence for the Macintosh computer. The availability of these software tools on personal computers is particularly important in that it will speed the porting to personal computers of the artificial intelligence development environments now running on more powerful systems. It will also facilitate the transfer of developed expert systems from these environments to run-on personal computers equipped with run-time inference engines.

Tools Products

"Tools" in this context encompasses software used for the development of end–user artificial intelligence software products. Artificial intelligence tools are in a sense midway between languages and end-user products, much as DBMS software occupies a niche in complexity and generality midway between such traditional programming languages as COBOL and such end-user business applications as general ledger. In PC software, two classes of tools exist today:

Natural language development tools

These aid in the development of natural language front ends to existing application software. One example is the Texas Instruments NLX Toolkit, which allows a programmer to de-

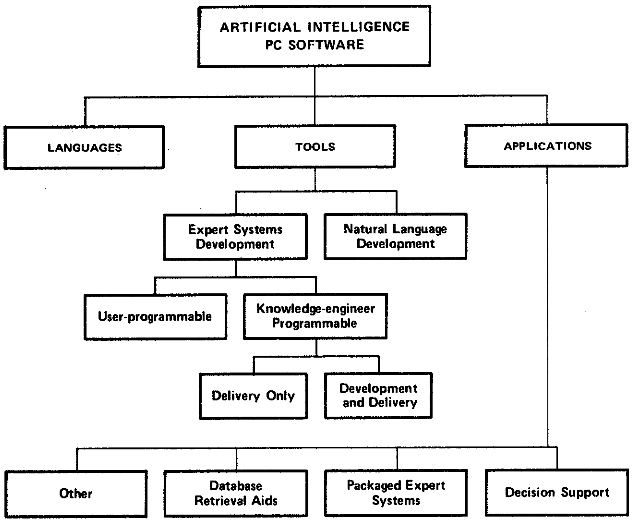


Figure 1—Today's artificial intelligence software for personal computers represented as a hierarchy of languages, tools, and applications

velop a sophisticated menu system (arguably not "true" AI). The user can then construct full, syntactically correct sentences, which are subsequently transformed into the command sequence expected by the underlying software product. An example of an end-user product demonstrating this capability is the Dow Jones Naturalink software package, also by Texas Instruments.

Other natural language development tools are presently under development. It is, however, a formidable task to develop a tool that is neither tied to a specific domain (e.g., database retrieval) nor restricted to a menu-style user interface. Consequently such products are unlikely to appear on the market for several years.

Expert systems development tools

These products, which are used to develop functional expert systems on the PC, represent a much larger and more diverse category than natural language development tools. Most of these were originally developed for more powerful systems and subsequently ported, in whole or in part, to personal computers. Obviously, the usefulness of the personal computer product depends in large measure on its pedigree. For example, M.1 by Teknowledge is based on the principles embodied in the high end S.1 system that Teknowledge also offers. On the other hand, the Personal Consultant of Texas Instruments is a PC embodiment of the older EMYCIN program, which some observers consider less sophisticated.

Both of the products mentioned above are complex and require considerable training to be used effectively. This limits their saleability to the business community, and, accordingly, efforts have been launched to develop expert system development tools which can be "trained" by the end user. One such tool, T.I.M.M. by General Research Corporation, allows the user to define the structure of the problem and then train the system by specifying the correct decision or inference under different starting conditions. The training examples specified by the user are then generalized by the software to develop a knowledge base that can be exercised to solve future occurrences. T.I.M.M. has been used to develop a number of full-scale commercial applications in the area; for example, of tuning the DEC VAX operating system, selecting sites for bank branches, and designing instrument clusters for helicopters. Another seemingly similar tool, Expert–Ease, developed at the University of Edinburgh, lacks this powerful generalization capability. Essentially, it is a tool for developing an efficient tree–structured search procedure for a limited knowledge base represented in a spreadsheet format.

In general, the end-user oriented tools are less versatile than their programmer oriented counterparts. The prospective user must evaluate this tradeoff in the context of his/her particular requirements.

At present, all expert systems development tools are both development and delivery vehicles; that is, they are used both for creating the knowledge base and for exercising it on a day-to-day basis. In the future, however, we can expect to see deliver-only software, which will accept a knowledge base developed on a specialized hardware/software environment (e.g., S.1, KEE, or ART), and operate it on the PC. Such a hybrid approach will allow the development of larger systems than can be constructed efficiently on the personal computer and will speed considerably the overall design and development process.

Applications

The applications software segment is perhaps the most turbulent and fascinating of all PC AI software. It contains several classes of product: natural-language database retrieval, decision support, packaged expert systems, and potentially a range of other products to come.

Natural-language database retrieval

Most users of personal computer database products find it difficult to master the techniques for retrieving data on an ad hoc basis, particularly when the needed data span several files. The first and so far the most successful company to address this problem is Microrim, with the Clout product line. Clout processes a database query typed in free-flowing English words of the users' choice and interrogates the corresponding files, which have been preprocessed to determine their structure, filed names, and contents. The product has programmed into it the syntax normally encountered in database queries and a vocabulary of key words, which users can extend. A related product, Savvy by Excalibur, also accomplishes database extraction but (at least in earlier versions) uses a pattern-matching technique less sophisticated and more vulnerable to error than the syntax processor in Clout.

Decision support

A promising application area for PC AI is the broad range of decision support software. A forefront product in this area is Lightyear, by the company of the same name. This product assists users in evaluating complex decisions involving tradeoffs among multiple quantitative and qualitative factors. The user specifies the range of outcomes and factors to be considered in a decision, then assigns a weight to each of them and a value to each pairing of factor and possible outcome, and finally creates a knowledge base to guide and further restrict the evaluation process.

Packaged expert systems

A range of software products now available carry particular areas of expertise to end users. Examples include the Negotiation Edge and the Sales Edge by Human Edge Software. Though intuitively appealing, such products need careful scrutiny. Some attempt to cover too broad an area of knowledge and as a result offer too little depth to be of significant value.

More promising in the short run are the expert systems believed now under development by some major American companies to provide employees with highly focused expert systems on low-cost delivery vehicles. Consider two potential examples: recommendation of financial products and services, to be used by bank staff in assisting their clients; and automotive repair systems, to be used by mechanics at auto dealerships. Though initially treated as proprietary products by the companies developing them, these capabilities will ultimately diffuse into the open market.

MARKET DRIVING FORCES

Developments in three primary areas—hardware, software, and design concepts—govern how quickly and in what direction PC AI develops. Of the three, hardware is probably least important, despite appearances to the contrary.

Hardware Forces

Improvements on the personal computer hardware front (e.g., the replacement of the IBM PC and PC-XT with the AT model) will probably have an evolutionary rather than revolutionary impact on the implementation of AI on personal computers. Processor speed no longer poses a critical limitation for many of the applications now in existence; in fact, the greater speed of the specialized AI computers is consumed to a great extent by their sophisticated display capabilities, not the inference engine. Many developers view these display capabilities as dispensable if necessary to port tools and applications to PCs.

Memory capacity is also decreasing as an obstacle to AI on PCs, as personal computers (and operating systems) reach the 1+ megabyte range. Moreover, the memory requirements of Lisp-based AI software is driven more by working set considerations tied to the interpretive nature of Lisp than to the size of the knowledge base or other application-related issues. As the software technology stabilizes, development tools will be ported to more conventional languages with much smaller working set requirements, decreasing the pressure on memory capacity.

Software Forces

As the above discussion has suggested, software development plays a key role in surmounting price/performance limitations that remain in hardware. The major areas for development are translating tools into nonsymbolic languages, finding less computationally intensive substitutes for the graphics techniques now applied, developing PC delivery vehicle counterparts for existing Lisp machine development systems, and finally improving the interfaces between AI software environments and non-AI software to which the "intelligence" is to be added (e.g., databases, operating systems, decision support tools).

Conceptual and Human Resource Drivers

One software developer described his company's strategy as to "cherry pick the best ideas of AI and apply them to a $2000 system." His statement suggests that perhaps the primary obstacle impeding the advance of PC AI software is a human one: the creativity required to identify fruitful application areas and the time required to design, code, test, and debug the software. A tremendous amount of experimentation is presently underway to determine how best to apply artificial intelligence techniques to existing software categories in ways which offer real value to the user. Just as Visicalc represented a breakthrough in conventional personal computer software—it was simple, powerful, and immediately useful despite its technical simplicity—similar discoveries must be made for the integration of AI into PC software to proceed.

SOME PROJECTIONS

Based on a review of the current state of PC AI software and the trends discussed above, it is possible to make some tentative predictions concerning this area of the industry. These are summarized in Figure 2.

From Tools to Applications

Whereas today the majority of the revenue in AI software products is generated by sales of tools, this will shift in the direction of applications as a broad range of PC AI applications develops. As has been the case in the mainframe and minicomputer industry, applications are typically of interest to a substantially larger market than are tools.

From Stand-Alone Products to Embedded Capabilities

Most of today's AI products are stand-alone in the sense that they use AI techniques in relatively pure form, with only weak integration with conventional software products and programming techniques. As the industry develops more experience and new concepts, this will shift until AI is more of a feature than a product in itself, much as computer graphics is now a feature of the user interface in many software products. When this happens, it will become quite difficult to account for the size (in dollars) of this segment of AI activity.

Several possible products can be envisioned; for example, an AI PC-DOS operating system that can process natural-

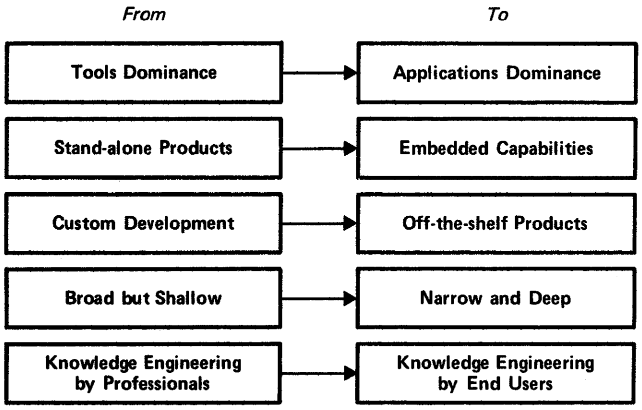


Figure 2—The migration of AI software to personal computers may transform the industry substantially

language requests for such operations as file copy and disk backup and apply a knowledge base to automate the simple but annoying tasks associated with system maintenance. Another fruitful application would be to apply expert systems techniques to analyze data generated by other applications; for example, an accounting package that would not only produce monthly financials but also analyze the numbers as a trained financial analyst would.

From Custom to Off-the-Shelf

Early on, we expect that much of the truly valuable PC AI software will be the result of funded proprietary development programs sponsored by major corporations. These programs will push the technology and provide valuable insight on how to develop practical cost-justified tools. As time goes on, these obvious early application areas will saturate, and the attentions of the software development industry will turn with greater success to the market for standardized products that can be sold off the shelf.

From Broad but Shallow to Narrow and Deep

Some of the early packaged expert systems for personal computers attempted to tackle too broad an area and hence do not perform convincingly. As the industry gains experience, it will follow the path of the large-system segment of the AI industry and focus on developing deeper, and hence more useful, applications in more limited domains. Unfortunately, the interim period may exacerbate the skepticism with which many now view the field.

From Knowledge Engineering by Professionals to Knowledge Engineering by End Users

The shortage of skilled knowledge engineers will lead to intensive efforts to develop better ways to develop expert systems without such assistance. This will take two forms: improvements in the capabilities of the present generation of end-user programmable tools, and improvements in the user friendliness of more sophisticated knowledge-engineer-oriented tools, ultimately to the extent of adding a natural-language interface to assist in the development of the knowledge base.

CONCLUSION

All projections are dangerous in a field as volatile as this one. However, it seems clear that as AI capabilities migrate to the personal computer, they have the potential to improve and transform the PC software industry in major ways. The full realization of this potential depends to some extent on improvements in hardware and software, but most notably on conceptual breakthroughs that have yet to occur.

Panel: Top executive view of data processing

Chair:
RAYMOND EPICH, *Northwest Industries,* Chicago, Illinois
Members:
EARL J. FREDERICK, *Children's Memorial Hospital,* Chicago, Illinois
DON HANSEN, *CFS Continental,* Chicago, Illinois
HOWARD LOVELY, *American National Bank & Trust Co.,* Chicago, Illinois
VICTOR VonSCHLEGEL, III, *Fibrex, Inc.,* Aurora, Illinois

In a recent study, 20 successful MIS heads listed their critical responsibilities. For this session, we asked the panelists to specify the criteria they use to evaluate their MIS functions. Their responses are compared to the list generated in the survey.

In addition to this question, the panelists also consider several key issues: Although the trend toward end-user computing has been well documented, who in the panelist's companies, besides the MIS staff, uses the computer directly? How widespread is the view that technology can be used to create a competitive weapon for companies? Can information technology be of strategic importance to their companies? These and other such questions are discussed.

Panel: Financial analyst view of the PC industry

Chair:
EDWARD CAWI, *Elgin School District,* Elgin, Illinois
Members:
ROBERT ARNOLD, *Robert Arnold & Associates,* Elgin, Illinois
ROBERT YOUNG, *R. J. Young & Associates,* New Lenox, Illinois

One of the major variables the financial analyst considers is the initial outlay. Price/performance considerations enter into the procurement decision as well as expectations of changes in the future. Another variable to consider is the continuing cost, which can be accelerated if a supplier discontinues a product line. The future viability of a supplier and hence the future costs of a personal computer product customer may depend on the product profile and on the research and development efforts of the supplier. This panel delves into these and other related issues.

Panel: The journalists look at the PC industry

Chair:
ARNOLD E. KELLER, *Hitchcock Publishing Co.,* Wheaton, Illinois
Members:
EDWARD J. BRIDE, *Software News,* Hudson, Massachusetts
JOHN B. DYKEMAN, *Modern Office Technology,* Cleveland, Ohio
WAYNE L. RHODES, *Infosystems,* Wheaton, Illinois

There is little doubt that the personal computer has brought about a revolution in information processing. As with any revolution, there are conflicting claims and considerable confusion. PC advertising is a combination of fact and fiction. Do the promises outweigh the performance? Are we creating islands of information with PCs? What are the bridges to the mainframe? Is the PC a productive office tool?

This session brings together a panel of expert journalists covering the world of information processing, discussing the todays and tomorrows of the PC revolution. The audience interacts with the journalists in a question-and-answer session.

Panel: Window management systems—fact or fiction?

Chair:
WILLIAM E. BRACKER, JR., *University of Arizona,* Tucson, Arizona
Members:
ARNOLD GREENFIELD, *University of Arizona,* Tucson, Arizona
ERNEST A. HERSHEY, *Database Design Inc.,* Ann Arbor, Michigan
BENN R. KONSYNSKI, III, *University of Arizona,* Tucson, Arizona

This session offers a taxonomy of current approaches to the use of windows in screen dialogues of application systems. Windows are viewed as a mechanism for managing presentation space in the design of information systems. Several neglected application areas are explored, including the use of multiple windows in the design of single applications. The potential and limitations of using windows is also discussed.

Panel: Artificial intelligence on personal computers

Chair:
DAVID R. BRODWIN, *Arthur D. Little, Inc.,* San Francisco, California
WAYNE J. ERICKSON, *Microrim, Inc.,* Bellevue, Washington
S. JERROLD KAPLAN, *Teknowledge, Inc.,* Palo Alto, California
WANDA RAPPAPORT, *General Research Corporation,* McLean, Virginia

In the past year, the capabilities of artificial intelligence have begun to appear in personal computer software products. This transfer of technology holds the potential to broaden the market for AI significantly and to transform the PC marketplace radically.

This session presents a framework for understanding and evaluating current PC-based AI software and discusses the hardware, software, and conceptual roadblocks to further developments. It will also project future trends in the converging arenas of AI and personal computing.

Panel: Popular portable computers

Chair:
RICK BAKER, *Hewlett Packard Company,* Corvallis, Oregon
Members:
NORMAN R. DeWITT, *Dataquest, Inc.,* San Jose, California
HELAYNE JONES, *Wang Laboratories,* Lowell, Massachusetts

This session is devoted to an analysis of popular portable computers. How popular are they? What features would make them more popular/useful? Although HP introduced the first portable computer that had the power of a desktop, was this type of system what the market wanted? Both Wang Laboratories and IBM are expected to enter the portable market. What will their approach be? These and other related issues are discussed.

Panel: Keyless data entry and automatic input technologies

Chair:
CARL T. HELMERS, JR., *North American Technology, Inc.,* Peterborough, New Hampshire
Members:
RUSS ADAMS, *Barcode News,* Peterborough, New Hampshire
DAVID CZAPLICKI, *Intermec,* Lynnwood, Washington
CONRAD J. KOTLOWSKI, *Department of Defense LOGMARS Coordination Group,* Tobyhanna, Pennsylvania

A driving force in applications of computers is the productivity improvement that comes from automation. The technologies of keyless data entry and automatic identification provide the link between physical items and databases. Applications include such diverse areas as file folders in the office, parts in manufacturing, inventory items, and express packages in transit. The members of this panel address contemporary bar code methods in commerical and military use as well as alternative methods of key entry bypass.

SOFTWARE SYSTEMS

C. ROBERT CARLSON, Track Chair
Illinois Institute of Technology
Chicago, Illinois

The THOR template editor

by KWANG-YA FANG
University of Illinois at Chicago
Chicago, Illinois

ABSTRACT

A form-entry system allows the user to view a database as a collection of data forms and therefore provides a user-friendly interface. Users can easily understand the information contained in forms and easily design forms of their own. The Template Handling On-line Reformatter (THOR) is an interactive software tool to support a form-entry system. The THOR system, designed for use at a terminal, provides a full-screen visual editor that allows the user to design different forms known as *menus*. To integrate THOR with a database system, the user may perform data-insertion, deletion, update, and retrieval through menu operations. This paper presents the general design of a prototype THOR system.

INTRODUCTION

The broad use of computers in different environments and the high cost of human resources in system development, maintenance, and application in recent years have redefined the definition of productivity. How fast a programmer can program is no longer a major concern of the data processing professional. Their attention is now focused on the *human* factors of computer systems[1,2] with considerations such as: *Is it easy to learn and understand? How little must the user do? How easy is it to do?*

Over the last 10 years, the psychological profile of the computer user has drawn the attention of many researchers in industry and academia. Many have begun to study database query languages from the end-user's point of view.[3-5] The motivation for such work has been diverse, but commonly reflects an attempt to define the notion of *ease of use* with some quantitative measurements. A survey of some of the experimental methods for evaluating query languages, and an elementary guide to how to read these kinds of experiments was recently presented by Reisner.[6] As Reisner points out, there is a general tendency to overstate conclusions, but it is obvious that a natural representation of data at the user's level is a major factor contributing to the *ease of use* of a database system. This paper briefly describes a form-entry system which allows the user to perform database operations via form manipulations.

Form is a common human communication media. Users can easily understand the information contained in forms and easily design forms of their own. A form-entry system allows the user to view a database as a collection of data forms. A data form from the user's point of view is merely a related set of data fields displayed in a desired format. The Template Handling On-line Reformatter (THOR) is an interactive software tool to support a form-entry system. The THOR system, designed for use at a terminal, provides a special screen editor that allows the user to perform database operations by manipulating forms known as *menus.* This database interface provides the user with a "natural" view of data which means that the user is no longer required to use conventional coding.

The THOR system consists of three software modules, the Interactive Template Entry Manager (ITEM), the Interactive Menu Editor (IME), and the Interactive Data Entry Administrator (IDEA). The general design of the THOR system is shown in Figure 1. ITEM defines the Template Definition File (TDF). TDF contains data fields to be included in a template and the information relating a template to the underlying database. IME enables the user to perform full-screen editing to generate the Menu Specification File (MSF) from a TDF or an existing MSF. The Menu Specification File contains visual and "navigation" information to display a logical template in a user-desired format. IDEA supports data entry and retrieval. It provides the user with a set of easy to use menu manipulation commands to perform data entry, modification, and deletion. A subsystem under IDEA, the Database Interface Subsystem (DBIS), supports database access operations. A prototype THOR system has been implemented in C language on a VAX machine running under the UNIX operating-system. The underlying database is currently a very primitive temporary system to demonstrate the operations of the THOR system. The general design of the THOR system is given in this paper with a brief description of the THOR interactive commands. Examples are given at the end of the paper to demonstrate the use of the THOR system.

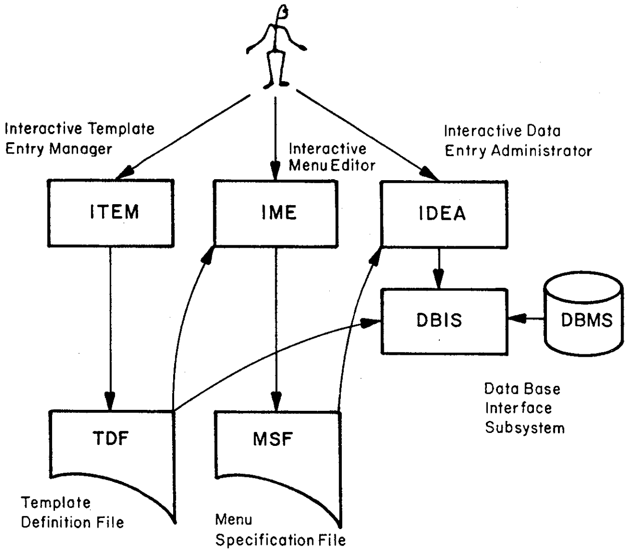

Interactive Template Entry Manager
Interactive Menu Editor
Interactive Data Entry Administrator
ITEM
IME
IDEA
DBIS
DBMS
Data Base Interface Subsystem
TDF
MSF
Template Definition File
Menu Specification File

Figure 1—The THOR form entry system

FIELD TYPES AND DEFINITIONS

A *logical template* is defined as a set of data fields with the necessary information to link each data field to the underlying database. A logical template may be associated with one or multiple data files. In general, a logical template consists of data fields to be filled in, and field names that indicate what information should be filled in the data fields. There are three types of fields currently supported by THOR:

1. *Primary field.* A single piece of information to be entered (i.e., *name*).

2. *Structure.* A composition of several different primary fields (i.e., the structure *date* may contain three primary fields: *month, day,* and *year*).
3. *List.* A set of identical rows (elements). Each element has one or more primary fields and/or structures (i.e., the list *student roster* would have the fields: *name, social security number, major, entering date,* and *grade point average.* The field *date* is a structure, and the others are primary fields).

Each field in a logical template consists of two parts, field name and field data. Field numbers are automatically assigned internally by the THOR system. To help the user access individual fields during a menu design session, the field numbers may be displayed on the screen adjacent to their associated fields at the user's option.

A menu is a logical template with visual and "navigation" information so that it can be displayed in a proper format. In other words, a menu is a logical template displayed in a user desired format. Each menu can be displayed on one or more screen images. Each screen image consists of data fields with associated name and text strings, plus some reserved spaces for terminal operator communications (help and error messages, etc.).

INTERACTIVE TEMPLATE ENTRY MANAGER (ITEM)

A logical template is defined by user commands in response to a series of prompts from the Interactive Template Entry Manager (ITEM). A Template Definition File (TDF) is created by ITEM to specify a logical template after the user has completed a template definition session. ITEM allows the user to create a new or modify an existing template definition file. When modifying an existing TDF, the updated logical template may be saved in a new file, or substituted for the original logical template in an existing TDF. Information collected by ITEM through these prompts include:

1. *Template contents.* Defines what data fields are included in a logical template.
2. *Data source.* Identifies the data files associated to the logical template.
3. *Data length.* Specifies the maximum length of each data field. If field length is not specified, the default value is the field length of the associated data field in the underlying database.
4. *Data format.* Selects one of two formats. Data fields in a logical template can be displayed in single- or multiple-line formats.

After collecting this information, ITEM performs certain database verifications against the underlying database directory so that queries to enter or retrieve data from the underlying database may be composed later by IDEA. For example, a key-field verification will ensure that all linked data files can be accessed.

In essence, a logical template is defined by ITEM with every field linked to the underlying database. A logical template, which may be further defined as a collection of data fields from one data record, a subset of one data record, or a conjunction of multiple data records or subrecords from different files, is independent of the logical or physical data layout of the underlying database. This allows the user to view the database as a collection of menus, and to perform all database interactions through menu operations.

INTERACTIVE MENU EDITOR (IME)

The Interactive Menu Editor (IME) is a full-screen visual editor designed specifically for menu editing. IME starts the menu editing session with an automatically generated default display by reading a Template Definition File (TDF) or a Menu Specification File (MSF). In IME-mode, the user is allowed to visually define or modify menus. Because a menu is a defined logical template displayed in a desired format, no new fields may be added in this mode. Therefore, only visual and navigation information are to be added in an MSF.

One screen image of a menu occupies the top 19 lines. The bottom five lines of the display screen are used by IME for command entry and message display. Commands are entered by simultaneously hitting a control key and a character key. If there is any associated argument which must be entered to complete the request, an argument requesting message is shown on the screen. When a command is executed, the screen display is immediately updated, and the command entry area is cleared for new commands. The immediate feedback of a command's effect on the display greatly reduces the effort required to design a menu, and makes editing easier to learn.

The menu editing commands are field-independent or single-field type commands. Field-independent commands are useful for setting global features in designing a menu. Normal full-screen operation commands are supported to allow the user to move screen images (page) forward and back, divide menus into pages, etc. Assistance commands display information that helps the user locate the position of each individual field (such as row and column coordinates), field numbers, and field positions. A field-selection command allows the user to select a target field for single-field operations.

The basic operation mode of the menu editor is field-at-a-time processing. Upon entry, IME starts an editing session in command-mode with all fields displayed. Normal editing operations are supported to allow the user to add, delete, modify, or replace characters and text. Single-field operations can be executed either by selecting a target field and performing the operation at the command line, or moving the cursor to the target position and performing the operation in place. A target field can be moved to any location on the menu by changing the coordinates of the field name.

INTERACTIVE DATA ENTRY ADMINISTRATOR (IDEA)

The Interactive Data Entry Administrator (IDEA) provides database interface that allows the user to perform data entry and retrieval at the menu level. IDEA operates at the user interface- and underlying database-interface levels. The design of IDEA at the user-interface level is very similar to that of IME and includes some special commands to support database operations. One basic difference between the two is that in data entry-mode, the user is restricted to manipulating field data, and is not allowed to change field names or move fields around; in menu editing-mode, the user cannot access field data.

Three different types of commands are available in IDEA, single-field, field-independent, and menu. Single-field and field-independent commands such as *search data, substitute data, change data,* or *enter data* provide data manipulation operations for individual fields. Also included are some commands specified in the menu editor which locate fields and move screen images. IDEA menu commands are for database operations such as inserting, deleting, updating, and reviewing menus.

Upon entry, IDEA displays a menu with no data. The bottom of the screen is reserved as the command-entry area. Data may then be loaded from the underlying database by a user specified menu key. This may be a single or a set of keys. If multiple keys are required, IDEA automatically requests that the user enter all key values. If a single key value is entered, an individual menu may be retrieved from the database. By specifying a range of key values, multiple menus may be retrieved for reviewing. The user may also perform data insertion, deletion, and updating through data entry commands in this mode.

The underlying database interface is performed by an independent subsystem, the Data Base Interface Subsystem (DBIS). Because DBIS is the only module which depends on the underlying database, THOR has the flexibility to be integrated into different database systems by rewriting or modifying of the DBIS subsystem.

For most database management systems (DBMS), different levels of abstractions of the database have been used to describe its design and application. A logical template or a menu can be viewed as another abstract representation of a database. To integrate THOR with a DBMS, the DBIS must be able to transform menus into lower-level database abstractions. The main concern is to transform a given form into the database conceptual schema, and vice-versa. Query language can then be generated automatically for data entry and retrieval.

Currently, a simple DBIS has been implemented interfacing with a very primitive database system to support and demonstrate the THOR system. In this prototype system, DBIS directly maps logical templates into the physical database. Because a logical template may be a composition of multiple underlying data records or subrecords, the DBIS first decomposes a logical template into "masks" that correspond to the associated data records. After the decomposition, each mask can be mapped in a one-to-one fashion to the underlying data record. If a mask contains only some of the fields from a record (i.e., a subrecord), only those fields in the mask will be included in the mapping. Conversely, mulitple data records may be mapped into masks and then merged together to form a logical template.

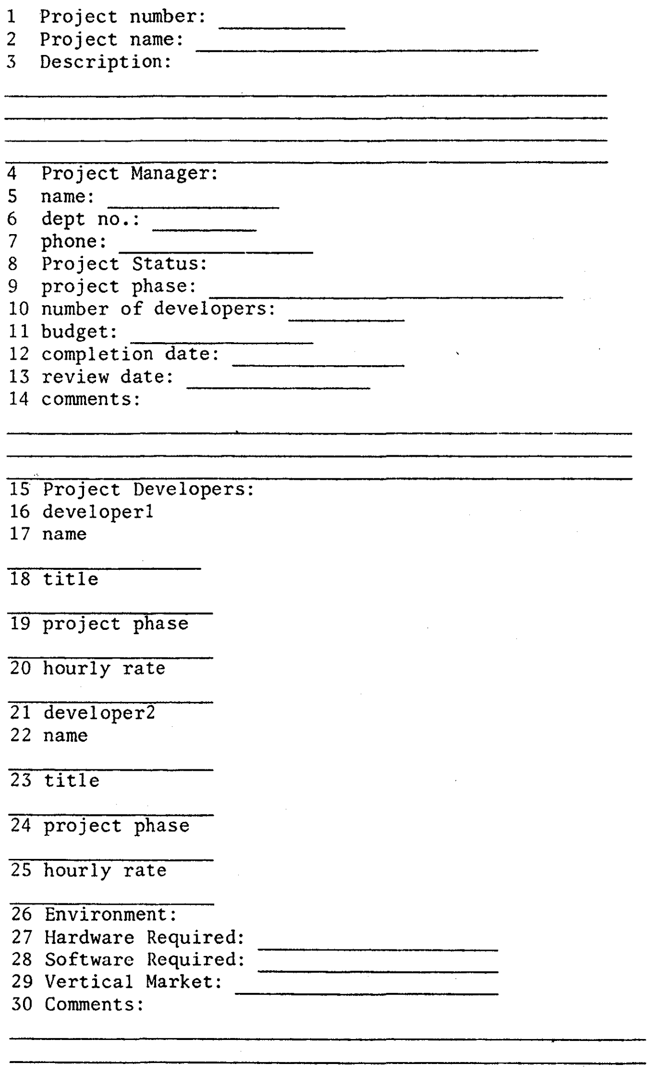
EXAMPLES

This section briefly demonstrates the use of the THOR system in interface with an underlying database. Some features and commands are not included.

Assume that a logical template has been defined in a TDF to track the status of a consulting company's software development project. Within the logical template, information relating to the project has been specified to include the following:

1. Primary fields—Project Number, Project Name, and Description.
2. Structure fields—Project Manager, Project Status, and Project Environment.
3. List field—Project Developers.

This logical template can be displayed in its default format when entering IME as shown in Figure 2. Note that each

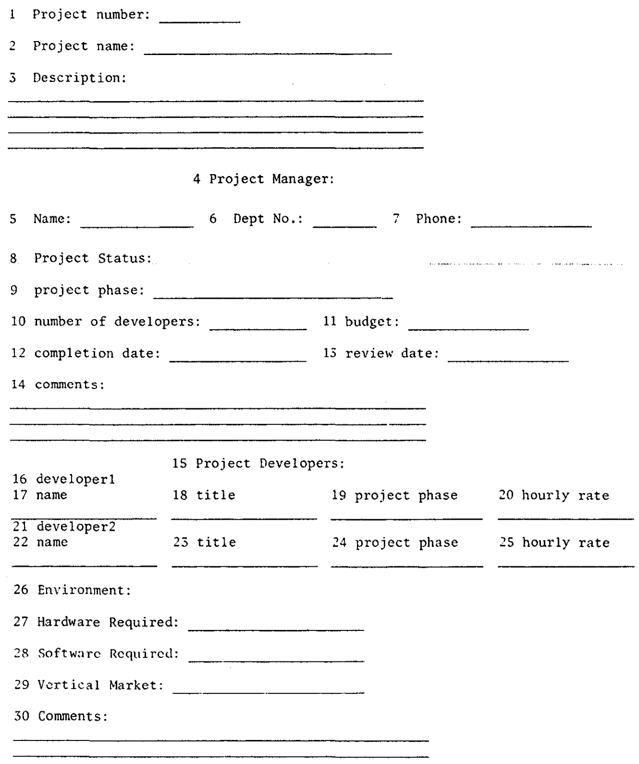


```
1  Project number: __________
2  Project name: ____________________________
3  Description:
_________________________________________________
_________________________________________________
_________________________________________________
_________________________________________________
4  Project Manager:
5  name: ______________
6  dept no.: _________
7  phone: _____________
8  Project Status:
9  project phase: ____________________________
10 number of developers: __________
11 budget: ______________
12 completion date: ______________
13 review date: _______________
14 comments:
___________________________________________________
___________________________________________________
___________________________________________________
15 Project Developers:
16 developer1
17 name
_______________
18 title
_________________
19 project phase
_________________
20 hourly rate
_________________
21 developer2
22 name
_________________
23 title
_________________
24 project phase
_________________
25 hourly rate
_________________
26 Environment:
27 Hardware Required: ____________________
28 Software Required: ____________________
29 Vertical Market: ______________________
30 Comments:
___________________________________________________
___________________________________________________
```

Figure 2—Automatically generated default menu

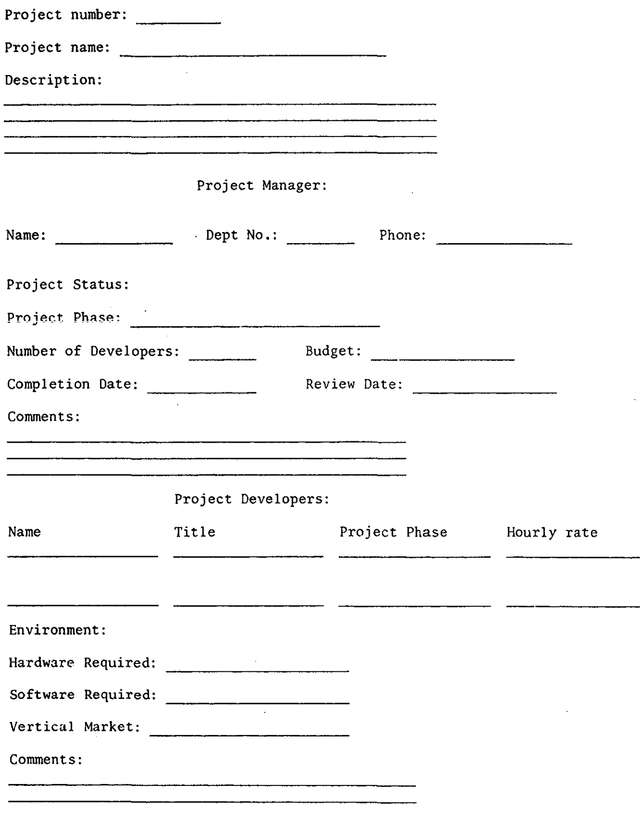
structure field is composed of several primary fields, and that the list field is a list of structures. For example, the structure *Project Status* (field 8) consists of the six primary fields *project phase, number of developers, budget, completion date, review date,* and *comments.* The list field *Project Developers* (field 15) contains two structures for two different developers; each such structure contains information fields for each developer (*name, title, project phase,* and *hourly rate*). This logic template includes the data files *project file* and *employee file.* The project file contains all information related to a project including project developers' names. The project developer's name also serves as the key field to the employee file. Therefore, information may be retrieved or entered by the project number through a project status logical template or menu.

The default menu display may be reformatted to a user desired format by moving fields around and changing field names. For example, the *Project Manager* field may be modified so that the field name is displayed in the center with three primary fields displayed on one line, as shown in Figure 3. Similarly, the list field *Project Developers* has been reformatted so that four primary fields are displayed on one line. In Figure 4, duplicated field names for each developer are removed and the menu is displayed in a more user-desired format. The field names in a menu may also be changed; they do not have to be identical to the names defined in the logic template. This gives the user the flexibility to select the field names in the display, and does not limit the user to field names as defined in the database.



```
1  Project number: _________

2  Project name: ____________________________

3  Description:
   _______________________________________________
   _______________________________________________
   _______________________________________________
   _______________________________________________

                   4 Project Manager:

5  Name: ____________   6 Dept No.: ______   7 Phone: ____________

8  Project Status:

9  project phase: ____________________________

10 number of developers: __________   11 budget: _____________

12 completion date: ______________   13 review date: _____________

14 comments:
   _______________________________________________
   _______________________________________________
   _______________________________________________

                   15 Project Developers:
16 developer1
17 name            18 title        19 project phase    20 hourly rate
_______________    ______________  ________________    ______________
21 developer2
22 name            23 title        24 project phase    25 hourly rate
_______________    ______________  ________________    ______________

26 Environment:

27 Hardware Required: ____________________

28 Software Required: ____________________

29 Vertical Market: ______________________

30 Comments:
   _______________________________________________
   _______________________________________________
```

Figure 3—Reformatted menu

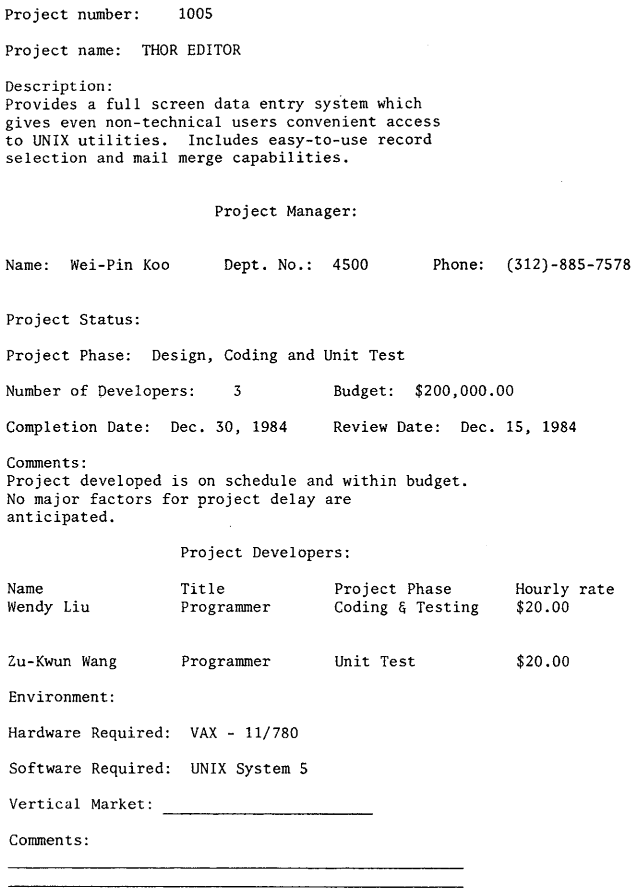


```
Project number: _________

Project name: ____________________________

Description:
_______________________________________________
_______________________________________________
_______________________________________________
_______________________________________________

                    Project Manager:

Name: ____________   Dept No.: _______   Phone: ______________

Project Status:

Project Phase: ____________________________

Number of Developers: _______      Budget: _______________

Completion Date: ___________      Review Date: ______________

Comments:
_______________________________________________
_______________________________________________
_______________________________________________

                    Project Developers:

Name             Title            Project Phase     Hourly rate
_______________  _______________  ________________  ______________


_______________  _______________  ________________  ______________

Environment:

Hardware Required: ____________________

Software Required: ____________________

Vertical Market: ______________________

Comments:
_______________________________________________
_______________________________________________
```

Figure 4—Final menu

After the user completes menu editing, a properly formatted menu may be saved in a Menu Specification File for future data entry operations. The menu specified in Figure 4, for example, can be saved with a menu name such as project[.spec].

With a defined menu, the user may enter data entry-mode by specifying the menu to be used. IDEA first displays the menu without data on the screen (as in Figure 4). The user may then perform data insertion, deletion, and update through data entry commands. To retrieve data, the user must first enter the value(s) for a key field(s) and then execute a *load data* command. In this example, Project Number is the menu key. Multiple menus may be retrieved by entering a range of key values (i.e., 1000 〈 .AND. 〈 1050). The user may insert data into the database by filling out the empty fields and executing *save data* command. The user must enter data in the key field(s), but is not required to enter data in every field in a menu for insertion. Each empty field is treated as data absent and is stored as an empty field in the database. A data update is done by first retrieving data from the database, changing or adding field data, and then executing a *save data* command to update the underlying database. In Figure 5, the project information for project number 1005 has been loaded into menu "project.spec" and displayed. Note that all internal field numbers are automatically removed when the final menu is displayed.



```
Project number:    1005

Project name:  THOR EDITOR

Description:
Provides a full screen data entry system which
gives even non-technical users convenient access
to UNIX utilities.  Includes easy-to-use record
selection and mail merge capabilities.

                         Project Manager:


Name:  Wei-Pin Koo      Dept. No.:  4500       Phone:  (312)-885-7578


Project Status:

Project Phase:  Design, Coding and Unit Test

Number of Developers:    3          Budget:  $200,000.00

Completion Date:  Dec. 30, 1984     Review Date:  Dec. 15, 1984

Comments:
Project developed is on schedule and within budget.
No major factors for project delay are
anticipated.

                      Project Developers:

Name             Title            Project Phase        Hourly rate
Wendy Liu        Programmer       Coding & Testing     $20.00


Zu-Kwun Wang     Programmer       Unit Test            $20.00

Environment:

Hardware Required:  VAX - 11/780

Software Required:  UNIX System 5

Vertical Market:  ____________________

Comments:
__________________________________________
__________________________________________
```

Figure 5—Load data into menu from database

CONCLUSION

The THOR form-entry system is a prototype developed in conjunction with a research project on the database operating environment. The principal objective is to design a database interface system in consideration of user behavior. Generally, users engage in goal-oriented activity; they always attempt to accomplish their goals as effortlessly as possible within the constraints imposed upon them.[5]

THOR is implemented to demonstrate user operations in data entry and retrieval. With a prototype system in place, user interaction and database operation via form manipulation can be studied. Further work in the transformation of forms into conventional query languages to integrate THOR with existing database systems is needed.

Different approaches have been proposed to make use of forms as an integral means of software application.[7–10] Most are concerned with the development of integrated office information systems and do not provide a general query language facility. The work of Embley's Form Processing System (FPS) introduces the notion of strong interpretation to map forms into relational algebra queries, thereby making it possible to extract data from a relational database and display it on a form as expected. In Embley's study, the transformation is concentrated on relational database systems. Further studies are needed to investigate the transformation for other models of database systems. An interesting and challenging work associated with form transformation is the study of performance enhancement, including query generation and optimization. The integration of other database features such as intra- and inter-form checking, and data verification should be studied. The goal is to develop a database operating environment emphasizing user behavior. The concept is to allow the user to interface with a database so that the user sees what a database is. As a result, minimal user knowledge about database and query language is required before the user can start using a database system. Because the user is performing database queries in a natural way of human thinking ("what you see is what you get"), it should give the user a high degree of confidence that his/her queries are correct. Furthermore, because in most cases there are no programs to write, it should be faster to perform queries in THOR system than in conventional query languages.

Many researchers believe that improvements in the relationship between computer systems and users will play a very important role in the development of computers. Difficulties in user interface has been the major obstacle to user acceptance of computer systems. The power of computers can be dramatically enhanced with a new type of software that provides a friendly operating environment and functions as an intermediary between computers and their users.

ACKNOWLEDGMENTS

The author would like to acknowledge W. Liu and W. P. Koo for their help in developing the prototype system.

REFERENCES

1. Ramsey, H. R. and Atwood, M. E. *Human Factors in Computer Systems: A Review of the Literature,* Science Applications, Inc., Englewood, Colorado, September 1979.
2. Moran, T. P. "An Applied Psychology of the User." *ACM Computing Surveys,* 13—1 (March 1981).
3. Thomas, J. C. "Psychological Issues in Database Management." *Proceedings of the 3rd International Conference on Very Large Data Bases.* Tokyo: October 1977.
4. Shneiderman, B. "Improving the Human Factors Aspect of Data Base Interactions." *ACM Transactions Database Systems.* 3—4 (December 1978).
5. Kim, W. "Relational Database Systems." *ACM Computing Surveys.* 11—3 (September 1979).
6. Reisner, P. "Human Factors Studies of Database Query Languages: A Survey and Assessment." *ACM Computing Survey.* 13—1 (March 1981).
7. Zloof, M. M. "A Language for Office and Business Automation." *Research Report RC8091.* IBM Research Laboratory, Yorktown Heights, New York, January 1980.
8. Luo, D. and Yao, S. B. "Form Operation by Example—A Language for Office Information Processing." *Proceedings of the International Conference on Management of Data.* Ann Arbor: May 1981.
9. Shu, N. C., Lum, V. Y., Tung, F. C., and Chang, C. L. "Specification of Forms Processing and Business Procedures for Office Automation." *Research Report RJ3040.* IBM Research Laboratory, San Jose, California, February 1981.
10. Embley, D. W. "A Forms Programming System." *Proceedings of the HICSS-15.* Honolulu: January 1982.

Structured application generation using XDB

by S. BING YAO and H. KITAGAWA*
University of Maryland
College Park, Maryland

ABSTRACT

A new approach for application system generation, based on the XDB application generator, is proposed. Combining functions of a relational database management system, interactive form processor, and sophisticated report writer, an approach that is based on a formal application model is developed. Advantages of such an approach include structured architecture, incremental development, dynamic prototyping, and high system modularity.

* H. Kitagawa is on leave from NEC Corporation, Tokyo, Japan.

1. INTRODUCTION

One of the most important problems involved in information system development is how to efficiently realize the required system functions. This topic has been studied for a long time under the central theme of *software engineering.* A considerable number of engineering disciplines have been proposed, and some of them are used in practical software projects. They include software development methodologies (e.g., top-down design and information hiding), specification techniques and tools (e.g., requirements/design specification languages and their processors, and high-level programming languages), development environments (e.g., programmer's work bench, and programming environments), and management disciplines (e.g., configuration control, software metrics, and cost estimation).[1–7]

Nevertheless, problems in software development, particularly in the development of application systems, are far from resolved. Many enterprises still encounter difficulties such as programs that do not meet the initial requirements, poor handling of requirements changes, and backlogs of undeveloped systems. The recent widespread use of microcomputers in business makes the situation even worse.[8] It brought about a tremendous increase in the number of computer utilization requirements, and with it the inevitability that nonprofessional programmers would have to develop application systems.

Recent attention has focused on a new approach for application system development using *application generator technology.*[9–13] With application generator techniques, the production of application systems is done without procedural coding and lengthy testing. Several application generators are available at present. They are classified from the following independent viewpoints.

1. Cross vs. resident target. A *cross* application generator generates application systems to be executed in a different target environment. This includes a different operating system or computer. Application systems generated by a *resident* application generator are to be executed in the same target environments.
2. Batch vs. interactive specification. A *batch* application generator generates application systems from specifications written in a specification language. This is similar in a way to the compilation of source programs into object codes. By contrast, an *interactive* application generator obtains system specification interactively through a dialogue of user-system interaction.
3. Program vs. interpretive code generation. A *program* generator generates source or object programs from application system specifications. An *interpretive* generator generates internal code or tables which are interpreted during execution.

An important advantage of application generators is that they can be applied to improve the software development life cycle. Application generators can be used for rapidly prototyping application systems.[7, 14] The prototype can be gradually augmented into the final system, or abandoned and the system developed in a conventional style. In either case, rapid prototyping is a very important technique for identifying and resolving problems in traditional software development.

In this paper, we discuss an approach for application generation using XDB.[15–17] XDB is an extended relational database system augmented by a set of application generation tools. In our classification, XDB is a *resident, interactive,* and *interpretive* application generator. It facilitates application generation through functions for relational database management, interactive form processing, sophisticated report generation, and automatic menu generation. XDB also enables application development based on a well-defined application system model. The model provides a basis for structured system design, and modeled components are directly generated with tools in XDB.

In Section 2, we describe the application system model. In Section 3, we discuss XDB functions which support the development of application systems representable within this model. XDB examples are given to illustrate the application generation process. Section 4 provides observations by comparing this approach with more conventional approaches. A summary is given in Section 5.

2. APPLICATION SYSTEM MODELING

In this section, we define a model of application systems under consideration within this context. This model specifies the range of application systems which can be generated with XDB.

As several studies in software engineering have clarified, an application system can be modeled from several viewpoints including functional hierarchy, state transition, and data flow.[18] In our model, an application system is first specified from the viewpoint of functional hierarchy. This corresponds to the hierarchical decomposition of the target system. A tree of functional components called *activities* results from this analysis (for example, see Figure 1). Upper level activities correspond to more abstract functions and lower level activities correspond to more primitive functions. In the decomposition here, leaf activities are to be represented as networks

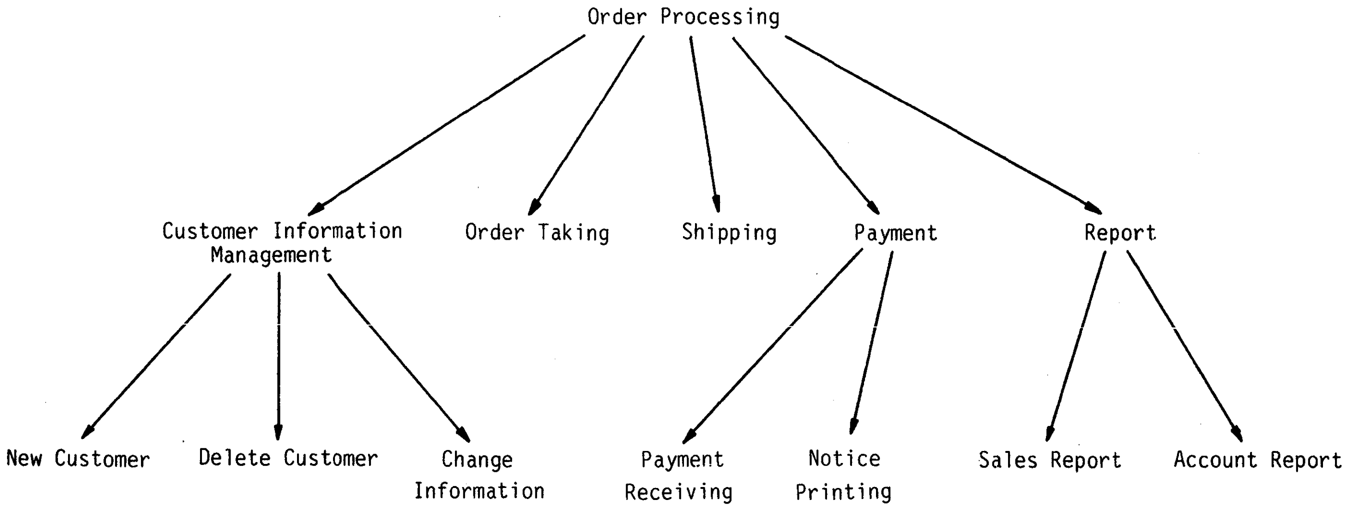


Figure 1—Activity tree

of transactions called *steps* (see Figure 2). Typical data manipulations involved in steps are data entry, database browsing, query, database update, and report generation. We now define the model in more precise terms.

An *application system* (ap) is modeled as the following tuple:

$$ap = (t = (A, de), D, tf, db),$$

where t is a functional hierarchy called an *activity tree*, denoted by t = (A, de), A is a set of functional modules called *activities*, and de: $A \rightarrow 2^A$ is an *activity decomposition function* meeting the *tree condition* defined below. An activity $a_i \in A$ is called a *unit activity* if $de(a_i) = 0$. D is a set of *activity diagrams*, tf: $U \rightarrow D$, where $U \subseteq A$ is the set of unit activities, is a function associating an activity diagram with each unit activity, and db is an application *database*.

Tree condition is defined as follows. Given (A, de), a *decomposition graph* can be created by representing each activity $a_i \in A$ by a node $n(a_i)$ and drawing a directed arc from $n(a_i)$ to $n(a_j)$ if $a_j \in de(a_i)$. If the decomposition graph forms a tree, (A, de) meets the tree condition.

Using the above formalism, we can represent a functional structure of an application system in an activity tree and associate a collection of primitive program modules with each unit activity. The control flow among the primitive program modules is represented in an activity diagram associated with it. Application menus are modeled as activity tree nodes.

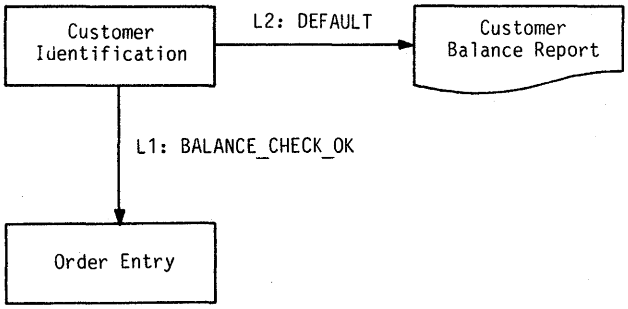


Figure 2—Activity diagram

Example 1

Figure 1 shows a sample decomposition graph of an activity tree where

A = {Order Processing, Customer Information Management, . . . , Sales Report, Account Report};
de (Order Processing) = {Customer Information Management, Order Taking, Shipping, Payment, Report};
de (Customer Information Management) = {New Customer, Delete Customer, Change Information}; and
de (Order Taking) = . . . , etc.

Note that a unit activity is represented by a leaf node in the decomposition graph. Here, we further define an activity diagram associated with a unit activity. An activity diagram $d_i \in D$ represents the control flow within a unit activity and is defined as the following tuple:

$$d_i \doteq (S_i, L_i, C_i, s_{io}, df_i),$$

where S_i is a set of primitive modules called *steps*, L_i is a set of *links*, C_i is a set of *conditions*, s_{io} is an element of S_i called a *start step*, and df_i is the following function associating two steps and a condition with a link,

$$df_i : L_i \rightarrow S_i \times S_i \times C_i .$$

An activity diagram can be represented in a graph in which nodes, directed arcs, and arc labels correspond to steps, links, and conditions, respectively. Each step $s_{ij} \in S_i$ corresponds to a primitive program module which, when executed, returns a *result value*. (The result value is similar to the return value of a procedure call, but steps are higher level objects than conventional procedures, and the result value is not restricted to a single value as in the return value of a procedure call.) Conditions to be associated with links by df_i are assertions on result values. When the activity diagram d_i is activated, start step s_{io} is first executed. After execution of step s_{ij}, the condition associated with every link joining out of s_{ij} is evaluated. Suppose the condition associated with link l_{ik} such that $df_i(l_{ik}) = (s_{ij}, s_{im}, c_{ik})$ is satisfied. Then the control is transferred to s_{im} and s_{im} is executed. A special DEFAULT condition is qualified only if none of the other conditions are satisfied. In this context, we assume the control transfer among steps is single thread and deterministic. In other words, we do not allow cases where more than two conditions are satisfied after the execution of a single step s_{ij}. If no conditions are met, execution of steps in d_i are terminated and d_i is inactivated. Working memory is provided during the activation of d_i. Steps in d_i can exchange intermediate results through the use of the working memory and the database.

Example 2

Figure 2 illustrates an example of an activity diagram where

S_i = {Customer Identification, Customer Balance Report, Order Entry};
L_i = {L1, L2};
C_i = {BALANCE_CHECK_OK, DEFAULT};
s_{io} = {Customer Identification};
df_i(L1) = (Customer Identification, Order Entry, BALANCE_CHECK_OK); and
df_i(L2) = (Customer Identification, Customer Balance Report, DEFAULT).

This activity diagram corresponds to the unit activity *Order Taking* in Figure 1. It has three steps. *Customer Identification* is a start step and is executed first. If the result value of the execution of this step is BALANCE_CHECK_OK, then the *Order Entry* step is executed. Otherwise, the *Customer Balance Report* step is executed after the execution of the start step.

So far, we have not specified detailed data manipulation performed in each step. In the context of XDB, application systems are developed around database manipulation and form-based interactive man-machine dialogue. The types of steps can be classified as follows.

1. Form processing
2. Report generation
3. Graph generation
4. Execution of a user's own program

In Figure 2, form processing steps (i.e., Customer Identification and Order Entry) are represented by rectangular symbols, and the report generation step (i.e., Customer Balance Report) is represented by a print-out symbol. We now further define each of these types of steps.

Step 1: Form processing. Form processing steps require interactive data manipulation utilizing formatted display templates on the screen. Forms are electronic version of paper forms so that they are user friendly in business data processing. Extensive research has been done using forms as the major means for man-machine interaction.[17, 19–29] Most operations required in business application systems such as data entry, data validation, database browsing, query, and database update are specified within the framework of form processing steps.

A *form processing step* (fp) is defined as follows:

$$fp = (t, C = BC \cup DC, V, D, sv, sd, P),$$

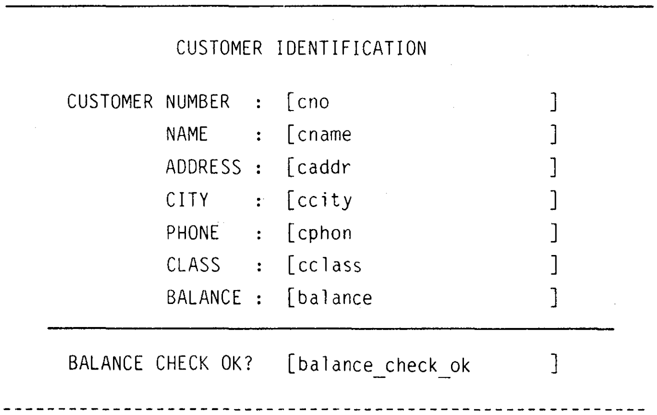
where t is a *form template* which determines the visual format of the form, C is a set of *cells* for entering and displaying data items, V is a set of *validation specifications,* D is a set of *derivation specifications,* $sv: BC \rightarrow V$ and $sd: DC \rightarrow D$ are *onto functions*, and P is a set of *post-processing specifications.* Cells are embedded into the form template. They may be defined either one by one as independent fields, or in a group constructing a table (with "repeating groups") or matrix. Cells are classified into mutually disjoint subsets, namely *basic cells* (BS) and *derived cells* (DS). They differ from each other as follows:

1. *Basic Cells:* Data items for basic cells are interactively entered by the user. Rules for validating input data items are given as validation specifications V and are associated with basic cells by function sv.
2. *Derived Cells:* Data items for derived cells are automatically retrieved and/or calculated from data in other cells, the database, and the working memory. For example, values for derived cells may be derived by the system function to display today's date, looked-up from the database, or calculated from other cell values. Rules for deriving the data are given as derivation specifications D and are associated with derived cells by function sd.

Validation specifications V can include the following constraints on input data items:

1. Type check
2. Value range check
3. Semantic consistency with other data values in
   (a) cells for the form processing step
   (b) the working memory
   (c) the database

Derivation specifications D give logic for deterministically deriving data values from other data values in the database, other cells, and the working memory using mathematical calculations, database look-ups, and some system functions (e.g., the function to get today's date). Post-processing specifications P specify operations performed when the user indicates the commitment for a form processing step. Normally, this happens after the user enters proper data values for all basic cells. Post-processing specifications P can include the following operations:



```
                  CUSTOMER IDENTIFICATION

CUSTOMER NUMBER  :  [cno                      ]
         NAME    :  [cname                    ]
         ADDRESS :  [caddr                    ]
         CITY    :  [ccity                    ]
         PHONE   :  [cphon                    ]
         CLASS   :  [cclass                   ]
         BALANCE :  [balance                  ]

BALANCE CHECK OK?   [balance_check_ok         ]
```

Figure 3—Customer identification

1. Computation of the result value
2. Write to the working memory
3. Update of the database

Data items displayed in cells are discarded after the form processing step is committed. Therefore, unless they are explicitly stored in the database or the working memory during post-processing, data entered by the user for basic cells are lost forever. (The system may provide a "form base"[22, 23] to directly store the complete data on forms. However, to simplify the model described here, we do not assume it in this context.)

Example 3

Figure 3 shows an example of a simple form template which might be used for the form processing step *Customer Identification.* Cells are represented by brackets. Suppose we have the customer information in the application database representable as the following table:

CUSTOMER (cno, cname, caddr, ccity, cphon, cclass, balance).

We can then designate cell "cno" as a basic cell, and make the others derived cells. That is to say,

BC = {cno}
DC = {cname, caddr, ccity, cphon, cclass, balance, balance_check_ok}

The validation specification for cell "cno" gives valid range of the data inputed for "cno" as well as its data type. It will also assert that the input value has to match exactly one occurrence for the attribute "cno" in the CUSTOMER table. The derivation specifications for the derived cells except "balance_check_ok" are basically queries to the table. The given "cno" value will be used for the key in the queries. The derivation specification for "balance_check_ok" will be a simple if-then-else logic. For example,

```
if cclass = 'A' and balance < LIMIT_FOR_A or
   cclass = 'B' and balance < LIMIT_FOR_B or
   cclass = 'C' and balance < LIMIT_FOR_C or
then balance_check_ok = 'OK'
else balance_check_ok = 'NO'.
```

Post-processing specifications will include two commands; the first gives a computation rule for the result value of this form processing step. It might be formulated as the following logic:

```
if balance_check_ok = 'OK'
then RESULT = 'BALANCE_CHECK_OK'
else RESULT = 'ERROR'.
```

The second command will be to write the current "cno" into the working memory provided for the activation of the *Order Taking* unit activity. This value is referred to in the execution of either *Order Entry* step or *Customer Balance Report* step to find out the "cno" of current concern.

Step 2: Report generation. Report generation is another important step in an activity. Essential data handling in a report generation step includes data retrieval from the database, calculation of derived data values, and formatted display and/or print-out. A form processing step can also formulate data retrieval, calculation, and formatted output. However, one of the major differences between form processing and report generation is that the former includes interactive data input from the user, while the latter does not. Another difference is that a small amount of data on a sheet of paper is manipulated in the execution of a form processing step. By contrast, the handling of voluminous data is not unusual in report generation.

Step 3: Graph generation. In general, graph generation is included in the same category as report generation. As in report generation, graph generation involves retrieval of relevant data, calculation, and formatted output of the result. However, the output is given as bar charts, line charts, pie charts, scatter plots, and so on.

Step 4: User program. Most business data processing can be formulated as form processing and report generation (including graph generation). However, since there are a variety of applications, in some cases all the user requirements for data processing cannot be met with the above three classes of steps. In such cases, the user can define a step as the execution of his own program. Of course, the user is provided with primitive program modules such as database interface routines to access the database.

3. APPLICATION SYSTEM DEVELOPMENT

As discussed in Section 1, XDB is a resident, interactive, and interpretive application generator. The architecture of XDB is shown in Figure 4. The use of XDB facilities is divided into two phases, *application system generation* and *application system execution.* In the former phase, XDB generates *applica-*

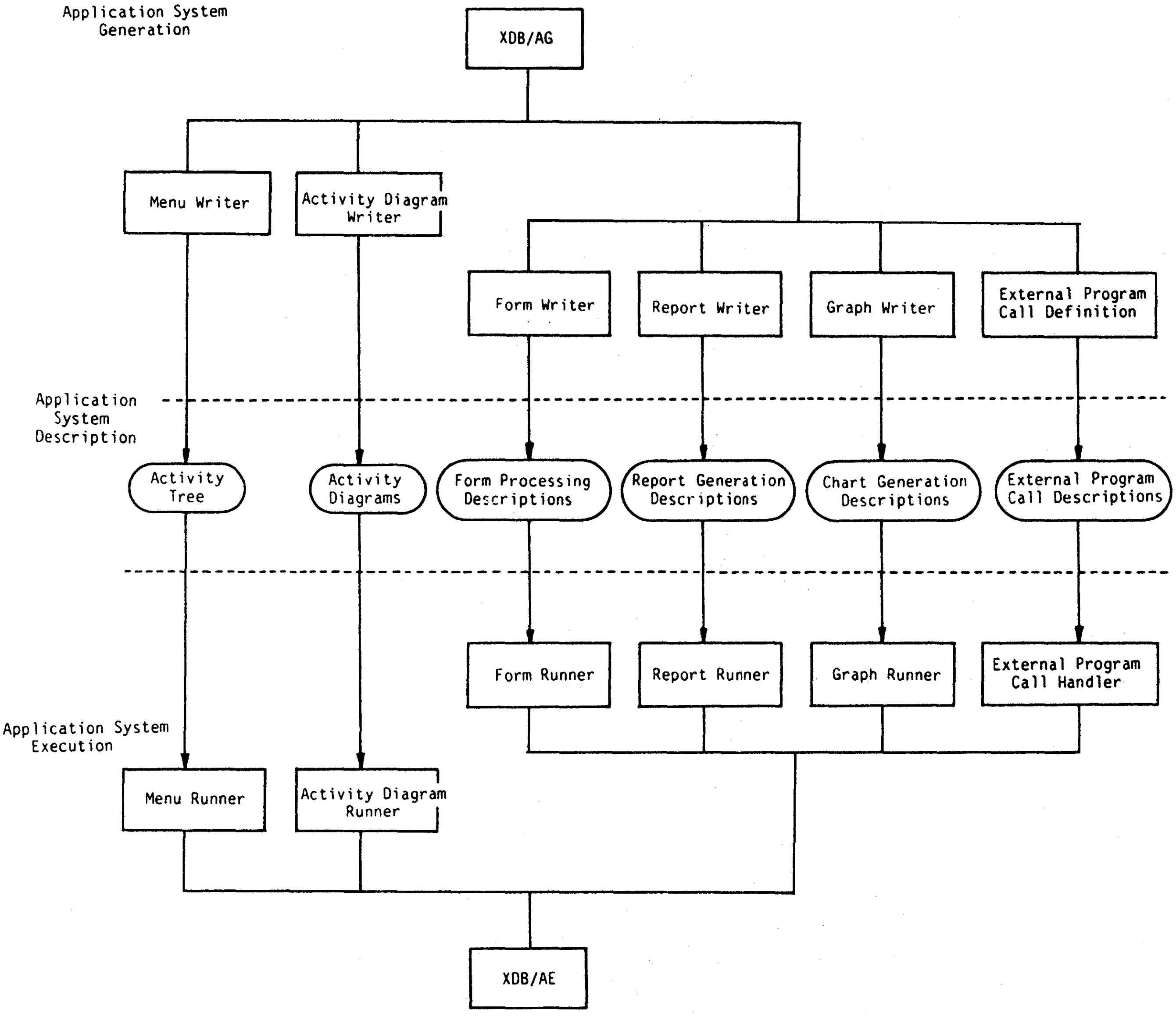

Application System Generation
XDB/AG
Menu Writer
Activity Diagram Writer
Form Writer
Report Writer
Graph Writer
External Program Call Definition
Application System Description
Activity Tree
Activity Diagrams
Form Processing Descriptions
Report Generation Descriptions
Chart Generation Descriptions
External Program Call Descriptions
Form Runner
Report Runner
Graph Runner
External Program Call Handler
Application System Execution
Menu Runner
Activity Diagram Runner
XDB/AE

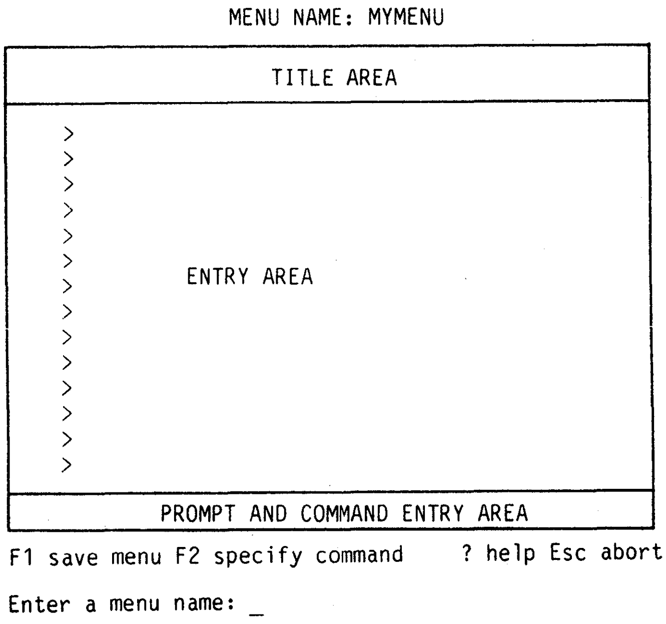
Figure 4—Architecture of XDB

tion system descriptions from the specifications given by the user. The user is provided with several interactive modules for activity tree definition and step definition. The application system descriptions are interpretively executed in XDB environments during the application system execution phase. Several types of interpreter modules are provided for the execution phase.

As the core for application development, XDB has the XQL relational database management system.[15,17] XQL's functions are based upon the specifications for the SQL/DS database management system.[30] In the XQL environments, XDB supports the interactive, evolutionary development of application systems. The following discussion shows how each aspect of the application system modeled in Section 2 is handled in XDB. We will use a small order processing system as an example. Suppose the requirements analysis and database design have already been done for the example system. We can define the following four tables.

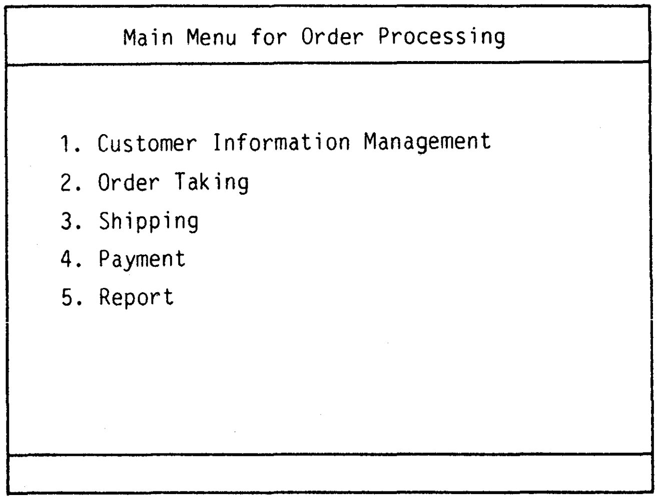
CUSTOMER (cno, cname, caddr, ccity, cphon, cclass, balance)
ORDER (ono, odate, cno, sname, saddr, scity, sphon, sdate, pdate)
PART-ORDER (ono, pno, pqty)
PART (pno, pname, pdesc, pprice, pqoh)

We will show how this kind of application system is interactively designed and generated.



```
                    MENU NAME: MYMENU
                       TITLE AREA
   >
   >
   >
   >
   >
   >
   >          ENTRY AREA
   >
   >
   >
   >
   >
   >
   >
              PROMPT AND COMMAND ENTRY AREA
F1 save menu F2 specify command      ? help Esc abort

Enter a menu name: _
```

Figure 5—Menu writer screen



```
          Main Menu for Order Processing

   1. Customer Information Management
   2. Order Taking
   3. Shipping
   4. Payment
   5. Report
```

Figure 6—Main menu

Activity tree

The development is performed in a top-down fashion. First, the activity tree is designed to determine the functional hierarchy of the target system. For this example, the activity tree as shown in Figure 1 is defined. In XDB environments, the activity tree is mapped to a hierarchical menu, and defined with the *menu writer.* Figure 5 shows a session example of the menu generation. No specification language is used for defining the menu. Instead, the definition is performed step-by-step through a number of interactive sessions. In these sessions, a set of commands can be used to specify menu screens corresponding to non-leaf nodes in the activity tree. The definition is completed through the command specification and the input of supplementary information, if required.

In the execution phase, the *menu runner* presents the menus. The menu screens defined in the generation phase are used to traverse the hierarchy and execute data processing operations formulated as unit activities. Figure 6 shows the main menu of the example order taking system. This menu corresponds to the root activity in the activity tree shown in Figure 1. When unit activities such as *Order Taking* and *Shipping* are selected, the steps defined in them are executed according to the flow to be specified in the activity diagrams. When other activities are selected from the main menu, a sub-menu is displayed on the screen.

Activity diagram

The activity diagram determines the control flow within a unit activity. With the use of the *activity diagram writer,* the user can interactively define the control flow structure. When the control structure is complicated, a visual aid with which the user can directly edit the activity diagrams becomes indispensable for performing the definition. Presently, XDB represents the control as a sequence of steps with all links tagged with condition NOT 'ERROR.' The definition is thus a linear list. If fatal errors occur in each step, the result value is automatically set to 'ERROR.' The result value 'ERROR' is also returned when the step is aborted upon user request or is set in post-processing.

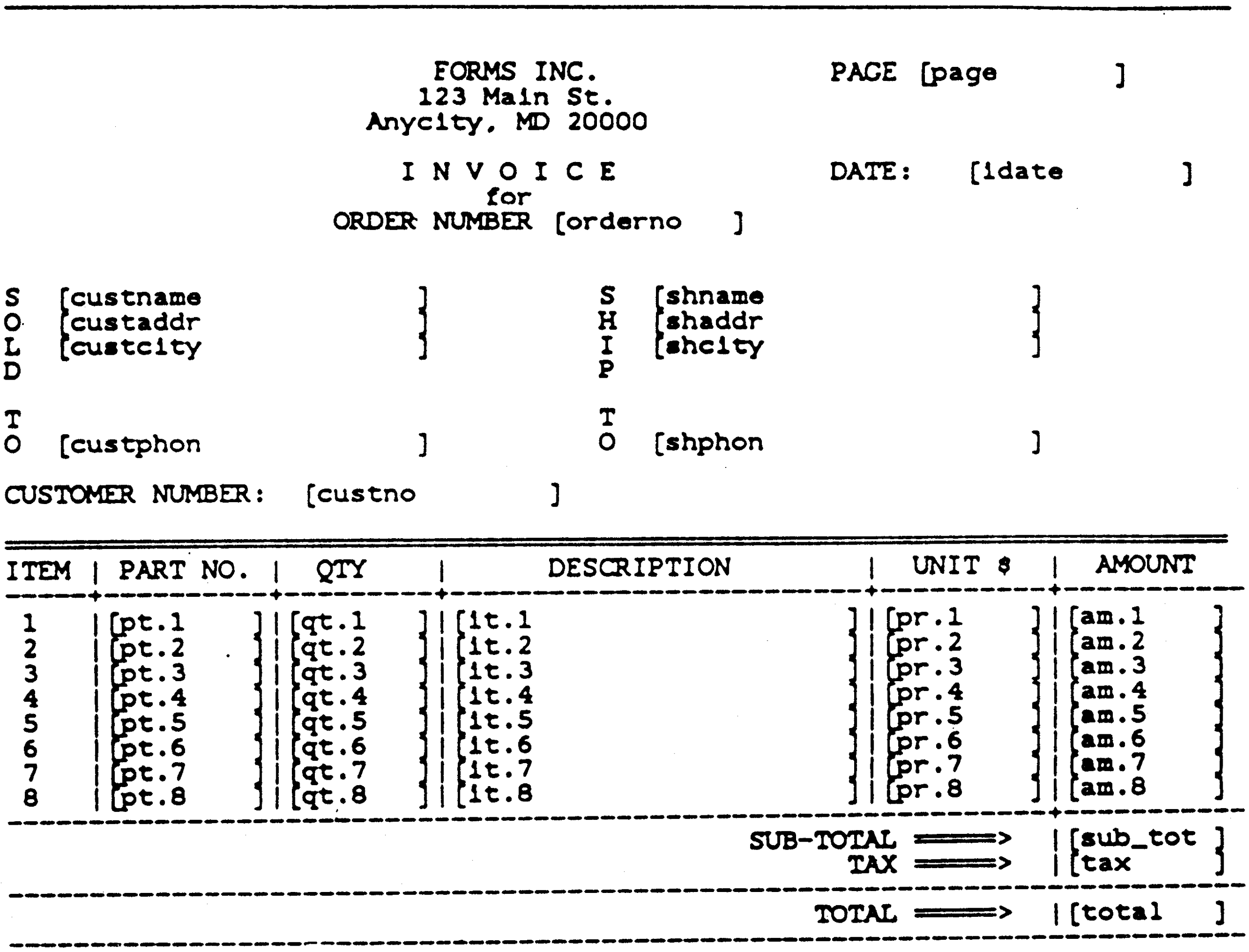
The definitions of step sequences are interpreted by the *activity diagram runner* to control the execution of steps in the execution phase. Suppose the activity diagram shown in Figure 2 is slightly modified here, and the sequence of steps *Order Entry* and *Customer Balance Report* is defined for the unit activity *Order Taking.* If *Order Taking* is selected from the main menu in Figure 6, then the step *Order Entry* is first executed. If its resultant value is 'ERROR', the execution of the activity is terminated. Otherwise, *Customer Balance Report* is executed as the next step to *Order Entry.*

Step

As mentioned in Section 2, four classes of processing are allowed for steps. In the above example of the activity *Order Taking, Order Entry* is a form processing step, and *Customer Balance Report* is a report generation step.

Form Processing. Form processing steps are interactively defined with the *form writer.* To define it, the user first designs the form template on the screen. In so doing, he also defines cells embedded in it. Cell type (i.e., basic/derived), data domain (i.e., integer/real/string/money/date), and display format information is associated with each cell. For basic cells, the user can give the validation specification, for example, indicating the range of input data values. For derived cells, he has to specify the derivation specification. The derivation specification is an expression combining arithmetic expressions and XQL query statements.

The form shown in Figure 7 will be used for *Order Entry.* As shown in this example, forms having complicated data



```
                         FORMS INC.                    PAGE [page        ]
                        123 Main St.
                      Anycity, MD 20000

                         I N V O I C E                 DATE:    [idate        ]
                              for
                    ORDER NUMBER [orderno   ]

S  [custname              ]      S  [shname              ]
O  [custaddr              ]      H  [shaddr              ]
L  [custcity              ]      I  [shcity              ]
D                                P

T                                T
O  [custphon              ]      O  [shphon              ]

CUSTOMER NUMBER:   [custno       ]

ITEM | PART NO.  | QTY     |        DESCRIPTION             |  UNIT $   |  AMOUNT
-----+-----------+---------+--------------------------------+-----------+------------
 1   | [pt.1    ]| [qt.1  ]| [it.1                         ]| [pr.1    ]| [am.1     ]
 2   | [pt.2    ]| [qt.2  ]| [it.2                         ]| [pr.2    ]| [am.2     ]
 3   | [pt.3    ]| [qt.3  ]| [it.3                         ]| [pr.3    ]| [am.3     ]
 4   | [pt.4    ]| [qt.4  ]| [it.4                         ]| [pr.4    ]| [am.4     ]
 5   | [pt.5    ]| [qt.5  ]| [it.5                         ]| [pr.5    ]| [am.5     ]
 6   | [pt.6    ]| [qt.6  ]| [it.6                         ]| [pr.6    ]| [am.6     ]
 7   | [pt.7    ]| [qt.7  ]| [it.7                         ]| [pr.7    ]| [am.7     ]
 8   | [pt.8    ]| [qt.8  ]| [it.8                         ]| [pr.8    ]| [am.8     ]
------------------------------------------------------------------------+------------
                                                   SUB-TOTAL =====>     | [sub_tot ]
                                                         TAX =====>     | [tax     ]
-------------------------------------------------------------------------------------
                                                       TOTAL =====>     | [total   ]
-------------------------------------------------------------------------------------
```

Figure 7—Order entry form

structures (e.g., repeating groups) can be defined in XDB. Here "custno," "pt.x," "qt.x," "shname," "shaddr," etc. are basic cells, and values for them are inputed by the user. Validation specifications could be given to assure the semantic constraints for maintaining proper balance values. Data values for cells "custname," "custaddr," "custcity," and "custphon" will be retrieved from the database. Their derivation specifications will be given as the following XQL query:

select cname, caddr, ccity, cphon
from CUSTOMER
where cno = [cno],

where "[cno]" means the input data for the cell "cno." Similarly, data values for "it.x" and "pr.x" for each table entry will be retrieved from the PART table. Derivation specifications for cells "am.x," "sub_total," "total," etc. will be based on arithmetic calculations from values in relevant cells. Data value for the cell "idata" for today's date will be derived by the built-in system function.

The post-processing specifications for this form processing will include storage requests of new records to tables ORDER and PART_ORDER. They are also specified using XQL database update statements. The balance value in the CUSTOMER table will also be updated during the post-processing.

Form execution is performed by the *form runner.* In the execution phase, data values for derived cells are obtained in a data-driven way; that is, their derivations are triggered as soon as their source data become available. Therefore, values for "sub_tot" and "total", for example, are automatically recalculated when a new data entry is given to the source cells involved with its computation. Data stored in the database is retrieved by directly executing the specified XQL statements embedded in the derivation specification. Commands in the post-processing specifications are executed in a similar way when the user indicates the commitment by pressing a proper function key.

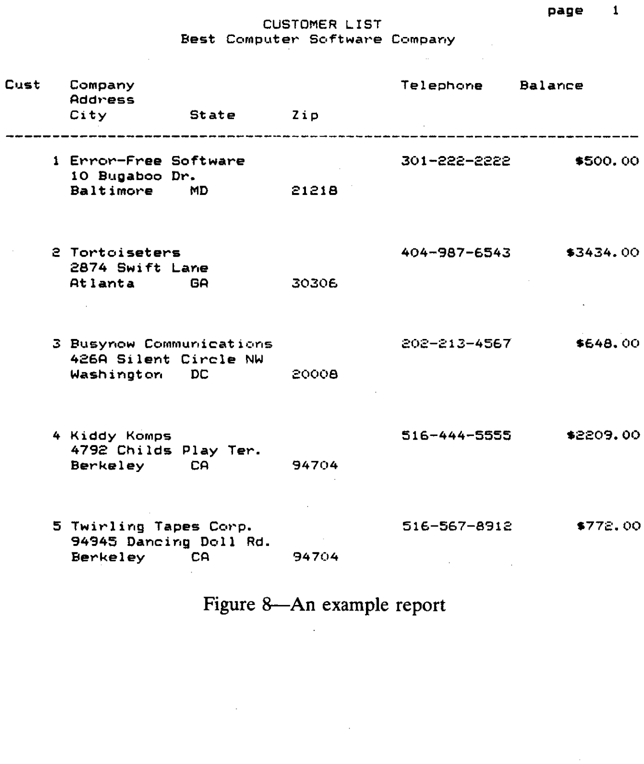
Report generation. Report generation steps are also defined interactively. In the generation phase, the user first formulates XQL queries and executes them to retrieve the data. In the report generation step, *Customer Balance Report,* the following XQL query will be used to retrieve the relevant data.

```
select cno, cname, caddr, ccity, cphon, balance
from CUSTOMER, ORDER
where CUSTOMER.cno < 6 and
      CUSTOMER.cno = ORDER.cno and
      odate < 1/1/84
```

The data is first displayed in a flat table on the screen, then the user interactively issues editing commands and formats the report into a desirable form. These are performed under the control of the *report writer*. The report writer automatically generates a description file that logs all of the interactive commands entered by the user. In the execution phase, the *report runner* interpretively executes the description to generate a fresh report from the current data stored in the database. Figure 8 shows an example report generated in the *Customer Balance Report* step.

Graph generation. Graph generation steps are defined and executed similarly to report generation, but differs in that the query results are shown in several types of graphs. XDB provides facilities for generating pie charts, bar charts, line charts, scatter charts and x–y plot.

User program. In principle, any program can be executed in steps in XDB environments to meet a variety of application system requirements. It can be a user-coded program or system program such as editor or utility programs. User-coded programs may be coded in languages such as C, BASIC, and Assembler. Typical examples of such user programs manipulate the database and are written in C. XDB provides the library for C programming language and facilitates the development of such programs. It includes basic functions for XQL relational database access and update. The return values of user programs become result values of the steps in which they are executed. User programs are pre-compiled outside XDB, and their object programs are incorporated in XDB environments under the *external program call definition module.* Their calls in the execution phase are handled by the *external program call handler.*



```
                                                   page    1
                      CUSTOMER LIST
              Best Computer Software Company

Cust   Company                              Telephone    Balance
       Address
       City          State        Zip
------------------------------------------------------------------

     1 Error-Free Software                  301-222-2222     $500.00
       10 Bugaboo Dr.
       Baltimore     MD           21218


     2 Tortoiseters                         404-987-6543    $3434.00
       2874 Swift Lane
       Atlanta       GA           30306


     3 Busynow Communications               202-213-4567     $648.00
       426A Silent Circle NW
       Washington    DC           20008


     4 Kiddy Komps                          516-444-5555    $2209.00
       4792 Childs Play Ter.
       Berkeley      CA           94704


     5 Twirling Tapes Corp.                 516-567-8912     $772.00
       94945 Dancing Doll Rd.
       Berkeley      CA           94704
```

Figure 8—An example report

4. ADVANTAGES OF THE XDB APPROACH

In Section 3, we explained application development in XDB. We now summarize the advantages of our approach over other application system development approaches.

Well-defined Application Architecture

Our approach provides a simple, yet powerful application model. The development along the model is directly supported by the utilities in XDB. The application system architecture is composed of well-defined components and is very modular. Therefore, the generated application system is easy to understand and has good maintainability. The development process can also be well-structured, since the target of the efforts is well specified.

Incremental Development

In the XDB environments, an application system can be developed incrementally from the prototype. For example, when only the activity tree is defined, the developer can evaluate the "shell" design by executing the menu handler. This will show how the hierarchical menus are presented to the user. After the evaluation and modification of the activity tree, the developer can proceed to the design of an activity diagram for each unit activity. In this way, the developer can gradually refine the prototype through the cycle of design and evaluation to obtain the final application system.

Dynamic Prototyping

Although the development can be gradually expanded from the core prototype, it need not follow a predefined specific development procedure. In some cases, the design of a specific step is of great importance and has an influence on the global architecture of an application system. In such cases, prototyping of the step of specific concern can be done even though the developer does not define the activity tree or activity diagrams. In this way, prototyping can be performed dynamically.

Modularity

Even though the development is performed incrementally and dynamically, modularity of the target system is always ensured within the XDB environments. The development process is performed in a top-down fashion, and the designer is guided into a structured design procedure which results in a more modular system. The target system is easy to test and modify.

Non-procedural Specification

The specifications of the target application system are interactively given by the developer in a non-procedural way. This not only facilitates the development, but enhances the maintainability of the target system because the specifications are given in high level descriptions. To modify or change the system, an analyst can interactively modify the design specification, and the system is automatically modified without the need for "system regeneration." Automatic generation of system documents can also be accomplished easily because the high level system descriptions are stored in the system in a well-organized way.

5. SUMMARY

In this paper, application generation with XDB is discussed. We proposed an application system model defining a class of applications under the XDB environments. Then, we discussed how each modeled aspect of the application system can be developed by XDB facilities. The discussion follows an example for a small order processing system. Finally, we summarized advantages of our approach over other system development approaches. The current version of XDB is a proprietary product implemented by Software Systems Technology on IBM personal computers. Versions that run on the generic MS/DOS and UNIX environments are also being developed. The high modularity of the system enables it to run with only 192K bytes of memory. The power and usefulness of the system have been verified through a number of experimental uses of XDB in realistic, personal-computer-based business application systems.

ACKNOWLEDGMENT

The authors are grateful to Mr. O. Q. Wu for reading and commenting on the draft of this paper.

REFERENCES

1. Basili, V. R. *Tutorial on Models and Metrics for Software Management and Engineering.* Silver Spring, Md.: IEEE Computer Society, 1980.
2. Boehm, B. W. "Software Engineering." *IEEE Transactions on Computers,* 25-12 (1976), pp. 1226–1241.
3. Bryan, W., C. Chadbourne, and S. Siegel. *Tutorial: Software Configuration Management.* Silver Spring, Md.: IEEE Computer Society, 1980.
4. Freeman, P. and A. I. Wasserman. *Tutorial: Software Design Techniques.* Silver Spring, Md.: IEEE Computer Society, 1983.
5. Putnum, L. H. *Tutorial: Software Cost Estimating and Life-cycle Control (Getting the Software Numbers).* Silver Spring, Md.: IEEE Computer Society, 1980.
6. Wasserman, A. I. "User Software Engineering and the Design of Interactive Systems." *Proceedings of the International Conference on Software Engineering,* San Diego, 1981, pp. 387–393.
7. Wasserman, A. I. *Tutorial: Interactive Development Environments.* Silver Spring, Md.: IEEE Computer Society, 1981.
8. Robinson, M. G. "SELECTOR IV: An Application Generator—On the Way to Becoming an Application System." *AFIPS, Personal Computing Digest,* 1981, pp. 260–267.
9. Cardenas, A. F. and W. P. Grafton. "Challenges and Requirements for New Application Generators." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 51), 1982, pp. 342–249.
10. Goodman, A. M. "Application Generators at IBM." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 51), 1982, pp. 360–362.
11. Grochow, J. M. "Application Generators: An Introduction." *AFIPS, Proceedings of the National Computer Conference,* (Vol. 51), 1982, pp. 390–392.
12. Horowitz, E., A. Kemper, and B. Narasimhan. "A Survey of Application Generators." *IEEE Software,* 2-1 (1985), pp. 40–54.
13. Martin, J. "Software for Application Development without Conventional Programming." *Software World,* 14-1 (1983), pp. 14–21.
14. Blum, B. I. "Rapid Prototyping of Information Management Systems." *ACM SIGSOFT Software Engineering Notes,* 7-5 (1982), pp. 35–38.
15. Liu, Y., D. Xia, and S. B. Yao. "Micro XDB—A Database Tool for Microcomputers." *Proceedings of the Technical Symposium of the Washington DC Chapter of ACM.* New York: Association for Computing Machinery, 1983.
16. Software Systems Technology. *XDB User's Manual (Version 1.0),* 1984.
17. Yao, S. B., A. R. Hevner, Z. Shi, and D. Luo. "FORMANAGER: An Office Forms Management System." *ACM Transactions on Office Information Systems,* 2-3 (1984), pp. 235–262.
18. Stephens, S. A., and L. L. Tripp. "Requirements Expression and Verification Aid." *Proceedings of the International Conference on Software Engineering,* Silver Spring, Md.: IEEE, 1978, pp. 101–108.
19. Czejdo, B. and D. W. Embley. "Office Form Definition and Processing Using a Relational Data Model." *Proceedings of the ACM SIGOA Conference on Office Information Systems,* New York: Association for Computing Machinery, 1984, pp. 123–131.
20. Fong, A. "A Model for Automatic Form-processing Procedures." *Proceedings of the 16th Annual Hawaii International Conference on System Science.*: Western Periodical Corporation, 1983.
21. Hammer, M., W. G. Howe, V. J. Kruskal, and I. Wladawsky. "A Very High Level Programming Language for Data Processing Applications." *Communications of the ACM,* 20-11 (1977), pp. 832–840.
22. Kitagawa, H., M. Gotoh, S. Misaki, and M. Azuma. "Form Document Management System SPECD OQ—Its Architecture and Implementation." *Proceedings of the ACM SIGOA Conference on Office Information Systems,* New York: Association for Computing Machinery, Toronto, 1984, pp. 132–142.
23. Kitagawa, H., T. L. Kunii, M. Azuma, and S. Misaki. "Formgraphics: A Form-based Graphics Architecture Providing a Database Workbench." *IEEE Computer Graphics and Applications,* 4-6 (1984), pp. 38–56.
24. Luo, D., and Yao, S. B. "Form Operation by Example—A Language for Office Information Processing." *Proceedings of the ACM SIGMOD Conference,* New York: Association for Computing Machinery, 1981, pp. 212–223.
25. Purvy, R., Farrell, J., and Klose, P. "The Design of STAR's Records Processing: Data Processing for the Noncomputer Professional." *ACM Transactions on Office Information Systems,* 1-1 (1983), pp. 3–24.
26. Rowe, L., and Shoens, K. "A Form Application Development System." *Proceedings of the ACM SIGMOD Conference.* New York: Association for Computing Machinery, 1982.
27. Shu, N., Lum, V., Tung, F., and Chang, C. "Specification of Forms Processing and Business Procedures for Office Automation." *IEEE Transactions on Software Engineering,* 8-5 (1982), pp. 499–512.
28. Tsichritzis, D. "Form Management." *Communications of the ACM,* 25-7 (1982), pp. 453–478.
29. Zloof, M. M. "Office-by-example: A Business Language that Unifies Data and Word Processing and Electronic Mail." *IBM Systems Journal,* 21-3 (1982), pp. 272–304.
30. Chamberlin, D., et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control." *IBM Research and Development,* 20-6 (1976).

Developments in dialog engineering

by MILDRED L. G. SHAW and BRIAN R. GAINES
York University
Toronto, Canada

ABSTRACT

The human-computer interface (HCI) is increasingly the major determinant of the success or failure of computer systems. It is time that foundations of dialog engineering for HCI were provided as explicit and well-founded as those for hardware and software engineering. HCI entered its period of theoretical consolidation at the beginning of the fifth generation in 1980. The lists of pragmatic dialog rules for HCI in the fourth generation have served their purpose, and effort should now be directed to the underlying foundations. Through the influences of other disciplines and their contribution to software engineering, a rich environment for HCI studies, theory, and applications now exists. This can be explored systematically by analyzing the various analogies to HCI possible when the parties are taken to be general systems, people, or computers. The fundamental principles at different levels may be used in the practical design of dialog shells for engineering effective dialog.

INTRODUCTION

This paper is concerned with techniques and underlying principles for *dialog engineering*;[1] that is, for programming effective human-computer interaction (HCI). Interest in HCI has existed since the early days of computing, became highly significant in applications as interactive computing came into widespread use in the 1960s, and is fundamental to fifth-generation computing system (FGCS) development. Users and commentators have been ambivalent towards the state of the art in HCI, oscillating between the euphoric prospects of human-computer symbiosis, and the frustratingly poor human-computer interfaces provided with many systems. An overview of the issues involved can be gained through a brief historic summary.

Computers are tools to enhance the capabilities of people. Human factor considerations have always played a role in computing system design. Babbage made extensive studies of logarithm table legibility with various combinations of type size, and ink and paper colors. Mauchly, in his 1947 discussion on the role of subroutine facilities in EDVAC programming, remarked that any machine coding system should be judged on how easy it is for the operator to obtain results.[2]

In the early years of computing, the machines were so slow, expensive, and unreliable that interactive use was rare in all but a few military and control applications. Those interacting with the machines were the few skilled operators who accepted the problems of interaction as minor compared to all the other difficulties of using computers. Professional ergonomic considerations of computer consoles date from Shackel's paper of 1959,[3] but it was the advent of timesharing systems in the early 1960s that made interactive access widely available and focused attention on human factors. In 1963/64 the MIT MAC,[4] RAND JOSS,[5] and Dartmouth College BASIC[6] systems pioneered a new style of computing whereby the problems of HCI became significant.

Timesharing systems opened computer use to a wider community that was less tolerant of technical problems. As early as 1967, Mills foresaw a future in which professionals would be among the least numerous and least significant system users.[7] White, in considering the problems of user communications said, "The user of many systems soon gets the feeling that either the messages he receives were hung onto the sytem as an afterthought or that they were designed for the convenience of the system rather than the user."[8] Unfortunately, this remark has not gone stale with age and is still echoed in the reviews of software packages.

In 1969, Nickerson surveyed work on man-machine interaction, remarked on its paucity, and noted the lack of contribution of psychology as a discipline to the design of interactive systems.[9] The problems created by poor HCI were highlighted by Walther and O'Neill at the NCC in 1974 where they noted the frustration of casual users with the unyielding, rigid, and intolerant dialog of time-sharing services, and its resultant damage to the industry.[10] Nickerson's 1981 paper on the same theme shows that many problems remain to be solved.[11]

The culmination of this emphasis on HCI may be seen in the Japanese announcement in 1981 of a program of development for a *fifth generation* of computing systems, and the funding of the ICOT research center in Tokyo.[12] Karatsu emphasized the HCI theme of this work by stating, "Until today, man had to walk toward the machine with effort to fit precisely. On the contrary, tomorrow, the machine itself will come to follow the order issued by man."[13]

At the start of the fifth generation forty years after the birth of computers, we know much about the human-computer interface, and we have many examples of good HCI, but we still produce systems that are extremely poor. Fortunately, the market place is coming to dominate the evolution of software and ensure the survival of only that with good HCI. Decreasing costs and the technologies of tomorrow will allow the complexity of the interface to increase, and the richness of multi-modal interaction through multi-media interfaces will place new demands on our HCI design capabilities.

It is time to provide foundations of dialog engineering for the human-computer interface as explicit and well-founded as those for hardware and software engineering. We have case histories and pragmatic rules, but not the theoretical framework within which the case histories can be understood and the rules developed. This paper outlines the basis for developing this framework during the next few years. The next section discusses how styles of dialog have been evolving. The following section systematically examines various sources of theoretical foundations. The final section discusses how these come together in the design of dialog shells.

STYLES OF INTERACTION

Hansen's comparison of the interactive system designer with a composer of music[14] appears very apt when one examines the wide variety of styles that have been developed for HCI. People and computers are very different systems; there are no universally obvious ways in which they should interact. Different designers have different views, often strongly held, and different users have different preferences, often equally strong. Three main styles of dialog may be distinguished.[15]

1. *Formal dialog,* in which the activities and data structures within the computer are presented externally in a direct

representation with the minimum syntactic sugar necessary to aid human cognition;

2. *Natural language dialog,* in which the human use of language to communicate information and commands is simulated within the narrow context of the activities and data structures within the computer;
3. *Graphic dialog,* in which the human manipulation of objects to communicate information and commands outside of the computer is simulated to access the activities and data structures within the computer.

The *prompt-response* dialogs of interactive JCLs such as those of VMS, Unix and CP/M are typical of formal dialogs. Zloof's *query-by-example* database retrieval system is a prime example of the simplicity and power of well-designed formal dialog.[16] It presents the structure of a relational database directly to the user with a minimum of syntactic sugar. The user can see what is there, the system is reasonably clear to naive users, and it is eminently natural to professional users.

There has always been ambivalence towards *natural language* dialog because it is remote from the actual operation of current computer systems.[17] Natural language is attractive because computer users already have the skills to use it to communicate with other people. However, we may risk that the very partial simulations of people that we can program currently are taken by users to have the full capabilities of people because they can "understand" natural language. In sociological terms, we may not be able to maintain the *role integrity** of a computer system with a natural language interface. In practice, commercial experience with such packages as Harris' *INTELLECT* indicate that they are attractive to users, and that problems of role integrity are not significant in applications to date.[18] It is probably more relevant to contrast applications where a concise formal dialog is adequate and more efficient than natural language with those where requirements are very open-ended and the complex formal language that might be required is best generated by translation from natural language. This is now feasible even on microcomputers using natural language shells such as *ASK.*[19]

Graphic dialog might more generally be termed *object simulation* because the graphic capabilities of the computer are used to create a model world that mimics the characteristics of the real world to the user. The screen of a Xerox *Star* simulates a desk on which are folders containing documents, in- and out-trays, and a trash bin.[20] Similar considerations apply to graphic dialog as to natural language dialog. Graphic dialog is attractive because it also brings into play the existing skills of a user, for example in dealing with an office environment and in the *direct manipulation*[21] of objects, but it becomes very important to ensure that all the user's expectations in that environment are properly fulfilled.

Within each of these styles of interaction, there has been a developmental sequence such that we can distinguish levels of capability. Figure 1 tabulates the development of styles of dialog from generations two through five of computing systems.

<table>
<tr><th></th><th>GRAPHIC DIALOG
Representing world</th><th>FORMAL DIALOG
Representing computer</th><th>NATURAL LANGUAGE DIALOG
Representing person</th></tr>
<tr><td>2
1956
-63</td><td>EXPENSIVE
Flight simulators
& process simulators
for limited military
& industrial use</td><td>SIMPLISTIC
Job control languages
giving operator access
to machine features</td><td>TRIVIAL
Natural language output
from stored script in
early computer-assisted
instruction.</td></tr>
<tr><td>3
1964
-71</td><td>PRACTICAL
Mimic diagrams.
Light pens.
Touch screens.</td><td>STANDARD
Simple prompt-response.
Menus.
Form-filling.</td><td>PRIMITIVE
Keyword recogntion.
Incorporation of words
input into output.</td></tr>
<tr><td>4
1972
-79</td><td>CREATING REALITY
Windows.
Icons.
Desktop simulation.</td><td>SOPHISTICATED
Dialog-engineered P-R.
Interactive F-F.
Intelligent F-F.</td><td>PRACTICAL
Understanding fixed domain.
Understanding DBMS domain.
Meta-level understanding.</td></tr>
<tr><td rowspan="6">5
1980
-87</td><td>LOW-COST
PC flight simulators.</td><td>INTEGRATIVE</td><td>SOPHISTICATED</td></tr>
<tr><td></td><td colspan="2">< * * * Integration through ASK and INTELLECT * * * ></td></tr>
<tr><td colspan="2">< * * * * * Integration through Macintosh * * * * * ></td><td></td></tr>
<tr><td colspan="2">< * * Integration through Symphony & Framework * * ></td><td></td></tr>
<tr><td colspan="3">< * * * * * * * * * * * * * * LISP machine applications * * * * * * * * * * * * * ></td></tr>
<tr><td colspan="3">< * * * * * * * * * * * * Fifth generation objectives * * * * * * * * * * * * ></td></tr>
</table>

Figure 1—Development of styles of interactive dialog

The center column shows formal dialog representing the computer characteristics directly developing from the early job control languages through simple prompt-response sequences suited to teleprinters; menus as visual displays became available; form-filling as cursor-addressable displays became available for use in transaction processing; dialog-engineered prompt-response with *HELP* facilities, default entries, and so on, as user support began to be considered; interactive form-filling as transaction processing became interactive; and intelligent form-filling with fields filled in through calculations as Visicalc was invented at the end of this era.

The left column shows graphic dialog developing from expensive applications in limited military and industrial systems through mimic diagrams, light pens and touch screens; the windows, icons and artificial reality of Smith's *Pygmalion*[22] which led to the Xerox *Star;* the decline in cost of computer hardware allowing flight simulation on personal computers; and a variety of integrated systems.

The right column shows natural language dialog developing from the trivial output of stored text in early computer-assisted instruction (CAI) systems; keyword recognition in later CAI; incorporation of input text in output as in Weizenbaum's *ELIZA*[23] and its practical application in Shaw's *PEGASUS;*[24] understanding a fixed domain as in Winograd's *SHRDLU;*[25] understanding a dynamic database as in Harris' *ROBOT*[26] (later *INTELLECT*); understanding the process of understanding itself as in Davis' TEIRESIAS;[27] and a variety of integrated systems.

We are now in the fifth generation era where the divisions between styles of dialog are beginning to break down as systems become increasingly integrated. The Apple *Macintosh* integrates graphic dialog in a simulated desk-top environment with the formal dialog of pull-down menus. Packages such as Lotus *Symphony* and Ashton-Tate's *Framework* integrate graphic and formal dialog in a variety of combinations. The natural language shells already mentioned, *ASK* and *INTELLECT,* translate natural user queries into formal DBMS enquiries. The advanced architecture of *LISP machines* supports high-resolution graphics, windows and icons, and is a natural

*Role integrity is consistency in maintaining the behavioral patterns associated by others with the role they perceive.

tool for work on all aspects of HCI including natural language. Such machines are the first examples of systems that begin to satisfy the FGCS objectives.

SYSTEMATIC ANALYSIS OF PRINCIPLES FOR EFFECTIVE DIALOG

As experience in the design and use of interactive systems grew, various authors presented guidelines for the design of effective human-computer dialog. Hansen at the 1971 FJCC seems to have been the first to develop user engineering principles for interactive systems.[14] The number of guidelines for dialog engineering has grown over the years. Ours started in 1975 as a set of 11 rules for programming interactive dialog,[28] growing to 30 in 1984.[15] They were first formulated for interaction through low-speed, upper-case teleprinters, but generalized well to visual displays and dialog through menus and forms. They appear to generalize adequately to dialog through windows, icons, and mice, and through restricted natural language. However, the proliferation of rules (a 1984 paper has over 500[29]), the development of new technologies for HCI, the growing maturity of studies of human cognition, human-computer interaction, and computer communication protocols, the need for standard dialog engineering tools, and the emergence of commercial products to satisfy that need suggests that the time is ripe for a new approach to dialog engineering.

All technologies go through phases after the initial breakthrough in which experience is gained, design rules are derived from experience, and theoretical foundations are derived to predict experience and derive design rules.[30] The breakthrough for HCI was the development of timesharing at the beginning of the third generation in 1964. Pragmatic rules were developed during the fourth generation from 1972 to 1979. During the fifth generation from 1980 to 1987, we should be seeing theoretical foundations develop for HCI. This theory must have its foundations in the larger context of studies of applied psychology, linguistics, computer communication, and general systemic principles. Although it may require fresh validation in the context of HCI, it has richer origins and support than studies of HCI alone. The following sections explore the wider context systematically.

System-system Interaction

One useful technique in analyzing HCI is generalization, noting that people and computers are both examples of *systems*. Systems can interact in many ways, but there are constraints on the type of interaction that we consider in HCI. In particular, we would expect that at least one of the systems will be *goal-seeking* and that the satisfaction of its goals, and the cost in doing so, will give us a basis for *evaluating* the interaction. The goal of one system may be to transfer information from the other through *communication,* to predict the behavior of the other through *modeling,* or to change the state of the other through *control.* Both systems may be goal-seeking, in which case considerations of *co-operation* and *competition* arise. The goal of one system may be to aid the other in, or to prevent it from communication, modeling or control.

There are many system-theoretic principles which are instantiated in the situation where one system is a person and the other a computer. Wiener emphasized this in his development of *cybernetics* as the study of "communication and control in men and machines."[31] Many principles of communication theory and control theory apply directly to HCI; for example, the dialog rule that *the user should dominate the computer*[1] is derived from a stability-theoretic result in control theory, that two coupled systems with similar time constants may oscillate unstably around their intended equilibrium state (i.e., the person modeling a computer and adapting to it while the computer is modeling the person and adapting to him is a potential source of mutual instability).

The system-theoretic foundations for dialog engineering are particularly important as technologies change. Concrete rules are needed that can be applied within a range of contexts without excessive development and inference, but when the context varies, the rule may become invalid. Indeed, the negation of a rule may become valid instead. The system-theoretic foundation for the rule is then essential in order to enable the variant appropriate to the new context to be derived. There are situations where it is appropriate for the computer to dominate the interaction, notably ones where the person is not able to adapt, but where the computer is able to do so to improve the interaction. An example might be the indexing of very large databases that a given person accesses infrequently and does not learn to navigate. The computer might then adapt its portrayal of the indexing material to the nature of the inquiry without ill-effects.[32]

Computer-system Interaction

If one of the interacting systems is taken to be a computer, then HCI can be seen to be analogous to interfacing a computer to another system such as a piece of equipment. The design principles applicable to computer-equipment interfaces are well known and carry over to person-computer dialog. Problems arise because the system to which the computer is to be interfaced already exists and is not another programmable computer. We may have to take it as it is and design an interface that copes with its peculiarities. The dialog rule *use the user's model*[1] derives from the idea that the dialog engineer should identify the existing interface and attempt to emulate it rather than change it. Problems also commonly arise through noise at the interface. The designer attempts both to provide a low-noise channel and to provide error-detection and correction for unavoidable noise. In HCI, such noise may arise through lack of clarity in information presentation giving rise to perceptual errors in one direction, mis-keying giving rise to errors in the other direction, and so on. The dialog rule *validate data on entry*[1] is a principle of communication over a noisy channel.

Computer-computer Interaction

When the interacting systems are both computers, it is possible to define *protocols* that may be implemented by any programmable digital system, and dialog engineering rules that may be seen as defining a *human protocol.*[33] The *Open System Interconnection* (OSI) ISO standard[34] is particularly interesting because it hierarchically structures computer-computer protocols for networks in a way that may have relevance for person-computer protocols. The concept of an *open system* is in itself relevant because it expresses objectives for computer networks that are equally applicable to people using those networks. The aim is to allow integrated systems to be formed from multiple components not all from one vendor and not all installed at the same time.[35] The OSI concept is that the network is open to all systems that conform in their communications with certain well-defined protocols. In human terms, the protocols may be seen as social norms for the behavior of members of a club; anyone may join provided they agree to conform to these norms.

The OSI standard is hierarchical, defining a number of layers, each with its own standard. Each of these levels has an analogy in human information processing and communication, e.g., the physical layer corresponds to the audio-visual perceptual processes, and there is much to be gained in applying the OSI concepts to the human protocol.[36]

Person-system Interaction

When one of the systems is a person we have the classic case of man-machine interaction. Considerations of people interacting with equipment has been treated as a branch of applied psychology termed *ergonomics* that arose under the same pressures as computer technology out of World War II studies of pilots, gunners, and so on. There is a wealth of results on general problems of human skills, training, its transfer between different learning situations, the effects of fatigue, and so on, that is immediately applicable to HCI. While interactive computers have been used primarily as programming and data entry systems, these effects have not been major considerations. However, as computer-based interfaces become increasingly the norm for a wide variety of human activities, the classic results of applied psychology and ergonomics are becoming increasingly important. The novelty of the computer should not blind us to its commonality with much earlier equipment. We do not have the time and effort to waste on rediscovering what is already known.

Person-person Interaction

When both systems are people, we have normal linguistic interaction from which the terms human-computer "conversation" and "dialog" have been generalized. Modern linguistic theory has become increasingly concerned with the interaction between participants in a dialog, rather than a view of linguistic output as a predefined stream to be decoded.[37] This provides a rich source of models for person-computer interaction, particularly as AI techniques take the computer closer to emulating people and their language behavior. There are also useful analogies to be drawn between the behavior of casual computer users and the transactional analysis of the behavior of strangers meeting.[38] Many of the principles of language such as *scoping* and *pronomial reference* are general principles of communication that occur in the graphic/gesture dialog with computers involving no textual interaction.[36]

DIALOG SHELLS

The considerations of the previous section suggest a hierarchy of levels of dialog rule generation (see Figure 2) including systemic principles, particular features of people and computers, considerations of their interaction, styles or technologies of interaction which lead to dialog shells, and particular applications.

The interaction level is interesting because it is where systemic person-and-computer principles come together. For example, the dialog rule *avoid acausality*[1] has a system-theoretic foundation in that causal modeling systems generate meaningless models of systems with even slight acausalities. However, to apply this principle, we have to know that people are causal modelers,[39] and we should not regard the rule as significant unless we know that time-sharing systems generate apparently random delay distributions.

At the instantiation level, the rules are applied to instances of styles and technologies of interaction. To make them application-independent, there has been a move to incorporate the rules in dialog shells that interface between the user and the computer system. The first general shells such as IBM EXEC originated from job control languages. A shell may incorporate many of the formal dialog rules for prompt-response.[15] Shells for form-filling transaction processing and menu-based systems are in widespread use. Several commercial companies have introduced window shells for the IBM PC, and Apple Macintosh has a general icon/mouse shell. *INTELLECT* is a natural language shell for mainframes, and *ASK* is one for micros. Expert system shells such as *EMYCIN* and *AL/X* are recent developments that incorporate signifi-

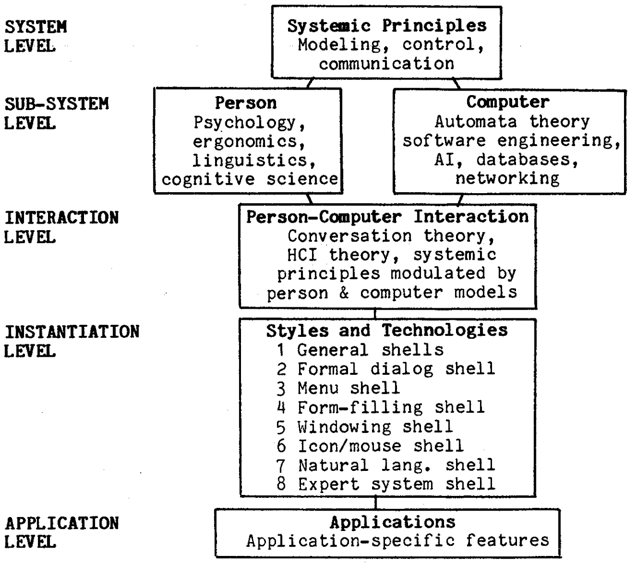

SYSTEM LEVEL
Systemic Principles
Modeling, control, communication
SUB-SYSTEM LEVEL
Person
Psychology, ergonomics, linguistics, cognitive science
Computer
Automata theory software engineering, AI, databases, networking
INTERACTION LEVEL
Person-Computer Interaction
Conversation theory, HCI theory, systemic principles modulated by person & computer models
INSTANTIATION LEVEL
Styles and Technologies
1 General shells
2 Formal dialog shell
3 Menu shell
4 Form-filling shell
5 Windowing shell
6 Icon/mouse shell
7 Natural lang. shell
8 Expert system shell
APPLICATION LEVEL
Applications
Application-specific features

Figure 2—Levels of dialog rule generation

cant new dialog techniques such as the uniform availability of *why* and *how* queries.[40]

CONCLUSIONS

The human-computer interface is increasingly the major determinant of the success or failure of computer systems. Its improvement is a major objective of the fifth generation computing development program. The literature from early days of computing to the present shows that the problems of HCI have been recognized but still adversely affect the use of computers. It is time that we provided foundations of dialog engineering for the human-computer interface as explicit and well-founded as those for hardware and software engineering. These may be based on the systemic, computational, psychological, and linguistic principles underlying HCI. They will then lead to the design of a variety of general-purpose dialog shells.

ACKNOWLEDGMENTS

Financial assistance for this work has been made available by the National Sciences and Engineering Research Council of Canada.

REFERENCES

1. Gaines, B. R. and M. L. G. Shaw. "Dialog Enegineering." in Sime, M., and M. J. Coombs (eds.), *Designing for Human-Computer Interaction*. London: Academic Press, 1983, pp. 23–53.
2. Mauchly, J. W. "Preparation of Problems for EDVAC-type Machines." in Randell, B. (ed.), *The Origins of Digital Computers: Selected Papers*. Berlin: Springer-Verlag, 1973.
3. Shackel, B. "Ergonomics for a Computer." *Design*, 120 (1959), pp. 36–39.
4. Fano, R. M. "The MAC System: A Progress Report." in Sass, M. A. and W. D. Wilkinson (eds.), *Computer Augmentation of Human Reasoning*. Washington: Spartan Books, 1965, pp. 131–150.
5. Shaw, J. C. "JOSS: Experience with an Experimental Computing Service for Users at Remote Consoles." in Orr, W. D. (ed.), *Conversational Computers*. New York: John Wiley, 1968, pp. 15–22.
6. Danver, J. H. and J. M. Nevison. "Secondary School Use of the Time-shared Computer at Dartmouth College." *AFIPS, Proceedings of Spring Joint Computer Conference* (Vol. 34), 1969, pp. 681–689.
7. Mills, R. G. "Man-machine Communication and Problem Solving." in Cuadra, C. A. (ed.), *Annual Reviews of Information Science and Technology, 2*. New York: Interscience, 1967.
8. White, R. R. "On-line Software—the Problems." in Gruenberger, F. (ed.), *The Transition to On-Line Computing*. Washington: Thompson, 1967, pp. 15–26.
9. Nickerson, R. S. "Man-computer Interaction: A Challenge for Human Factors Research." *IEEE Transactions on Man-Machine Systems*, MMS-10 (1969), 4, pp. 164–180.
10. Walther, G. H. and H. F. O'Neil. "On-line User-computer Interface—the Effects of Interface Flexibility, Terminal Type, and Experience on Performance." *AFIPS Proceedings of the National Computer Conference* (Vol. 43), 1974, pp. 379–384.
11. Nickerson, R. S. "Why Interactive Computer Systems are Sometimes Not Used by People Who Might Benefit by Them." *International Journal of Man-Machine Studies*, 15 (1981), 4, pp. 469–483.
12. Moto-Oka, T. (ed.), *Fifth Generation Computer Systems*. Amsterdam: North-Holland, 1982.
13. Karatsu, H. "What is Required of the Fifth Generation Computer—Social Needs and Impact." in Moto-oka, T. (ed.), *Fifth Generation Computer Systems*. Amsterdam: North-Holland, 1982, pp. 93–106.
14. Hansen, W. J. "User Engineering Principles for Interactive Systems." *AFIPS Proceedings of the Fall Joint Computer Conference* (Vol. 39), 1971, pp. 523–532.
15. Gaines, B. R. and M. L. G. Shaw. *The Art of Computer Conversation: A New Medium for Communication*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.
16. Zloof, M. M. "Office-by-example: A Business Language that Unifies Data and Word Processing and Electronic Mail." *IBM Systems Journal*, 21 (1982), 3, pp. 272–304.
17. Hill, I. D. "Natural Language Versus Computer Language." in Sime, M. and M. J.Coombs (eds.), *Designing for Human-Computer Interaction*. London: Academic Press, 1983, pp. 55–72.
18. Harris, L. R. "Experience with INTELLECT." *AI Magazine*, 5 (1984), 2, pp. 43–50.
19. Thompson, B. H., F. B. Thompson, and T.-P. Ho. "Knowledgeable Contexts for User Interaction." *AFIPS, Proceedings of the National Computer Conference* (Vol. 52), 1983, pp. 29–38.
20. Smith, D. C., C. Irby, R. Kimball, B. Verplank, and E. Harslem. "Designing the Star User Interface." In Degano, P., and E. Sandewall (eds.), *Integrated Interactive Computing Systems*. Amsterdam: North-Holland, 1983.
21. Shneiderman, B. "Direct Manipulation: A Step Beyond Programming Languages." *Computer*, 16 (1983), 8, pp. 57–69.
22. Smith, D. C. *Pygmalion*. Basel, Switzerland: Birkhauser, 1977.
23. Weizenbaum, J. *Computer Power and Human Reason: From Judgement to Calculation*. San Francisco: W. H. Freeman, 1976.
24. Shaw, M. L. G. *On Becoming a Personal Scientist*. London: Academic Press, 1980.
25. Winograd, T. *Undersanding Natural Language*. Edinburgh: Edinburgh University Press, 1972.
26. Harris, L. R. "User Oriented Data Base Query with the ROBOT Natural Language Query System." *IJMMS*, 9 (1977), 6, pp. 697–713.
27. Davis, R., and D. B. Lenat. *Knowledge-Based Systems in Artificial Intelligence*. New York: McGraw-Hill, 1982.
28. Gaines, B. R., and P. V. Facey. "Some Experience in Interactive System Development and Application." *IEEE Proceedings*, 63 (1975), pp. 155–169.
29. Williges, B. H., and R. C. Williges. "Dialogue Design Considerations for Interactive Computer Systems." in Muckler, F. A. (ed.), *Human Factors Review: 1984*. Santa Monica: Human Factors Society, 1984, pp. 167–208.
30. Gaines, B. R., and M. L. G. Shaw. "Generations of Computers: Modeling and Forecasting." *Possible Worlds*. 1–4 (1985), pp. 3–16.
31. Wiener, N. *Cybernetics*. Cambridge, Mass.: MIT Press, 1948.
32. Witten, I. H., S. Greenberg, and J. Cleary. "Personalizable Directories: A Case Study in Automatic User Modelling." *Proceedings of Graphics Interface '83*. Ottawa: National Research Council of Canada (1983), pp. 183–189.
33. Shaw, M. L. G., and B. R. Gaines. "Does the Human Component in a Network Have a Protocol?" *Proceedings of the International Electrical, Electronics Conference and Exposition*. IEEE 83CH1955-4 (1983), pp. 546–549.
34. Day, J. D., and H. Zimmerman. "The OSI Reference Model." *IEEE Proceedings*, 71 (1983), 12, pp. 1334–1340.
35. Gaines, B. R. "From Word Processing to Image Processing in Office Systems." *Proceedings of the International Electrical, Electronics Conference and Exposition*. IEEE 83CH1955-4 (1983), pp. 622–625.
36. Taylor, M. M., C. A. McCann, and M. I. Tuori. "The Interactive Spatial Information System." *DCIEM Report 84-R-22*. Toronto: DCIEM, 1984.
37. Bennett, J. *Linguistic Behaviour*. Cambridge, UK: Cambridge University Press, 1976.
38. Berne, E. *What Do You Do After You Say Hello?* London: Andre Deutsch, 1974.
39. Gaines, B. R. "On the Complexity of Causal Models." *IEEE Transactions on Systems, Man & Cybernetics*. SMC-6 (1976), 1, pp. 56–59.
40. Gevarter, W. B. "Expert Systems: Limited but Powerful." *IEEE Spectrum*, 18 (1983), pp. 39–45.

Software productivity and its management

by JAINENDRA K. NAVLAKHA
Florida International University
Miami, Florida

ABSTRACT

Software productivity is one of the most important attributes of the software development process. Our current knowledge of software productivity is not sufficient to define it properly, let alone measure it accurately. Indeed, for lack of something better, the industry has used some version or other of *lines of code per man-month* as the measure of productivity. To my mind, this measures programmer productivity, not software productivity. In this paper, I discuss several possible definitions of software productivity from different viewpoints. I also discuss an important issue on the subject, the management of software productivity.

INTRODUCTION

Software engineering refers to the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.[1] The annual cost of software in the United States, which was approximately $40 billion in 1980[1] or about 2% of GNP, is estimated to grow to 13% of GNP by the year 1990.[2] Over the past two decades, the cost of hardware has steadily decreased while the cost of developing software has consistently increased. The demand for software is increasing faster than our ability to supply it.

In such an economic atmosphere, it is necessary that methods be developed to measure various attributes of computer software in order to continually enhance our ability to increase productivity. The sector of software engineering that deals with these measurements is called *software metrics*. Software metrics has evolved from the gradual recognition of the need and importance of measuring and controlling the quality and development cost of software product.

Measurement discipline is as fundamental to programming as it is to any other area of engineering. Boehm, et al. define a software metric as a measurement of the extent or degree to which a software product possesses and exhibits a certain quality, property, or attribute.[3] Software metrics can be used to measure many properties and attributes of software including reliability, complexity, productivity, quality, correctness, availability, maintainability, portability, and development effort. They can also be used to provide a precise definition of certain measurements to be used in legal contracts, to manage resources, and most of all to evaluate the quality of a design and pinpoint potential problem areas of that design so that changes and improvements can be made at any stage during the software development process.

Thus, it is evident that software metrics is a very important area of software engineering. However, it is also one of the most difficult areas of research. As Munson and Yeh report, "... the definition of the attributes of software which materially affect cost, quality and productivity are still unknown, or when known, are unmeasurable. Until we can measure and compare we cannot consider computer science a science."[4]

One of the most important attributes of the software development process that needs to be measured is software productivity. In most software houses, productivity has been measured as follows:

$$\frac{\text{Total deliverable source instructions}}{\text{Number of man-months to develop and correct}}$$

A slight variation of the above definition of the form of number of modules in the software per unit time period has also been used by some. This variation essentially reduces to the original definition by approximating the number of source lines per module, assuming proper modularization.

Note that the above mathematical definition does not truly define software productivity as much as it defines programmer productivity. While it is quite important to have a measure for programmer productivity, this is not the subject of this paper. Hence, I will stay away from the above definition. It is quite conceivable that until a better measure for software productivity is available, most people will continue to use programmer productivity instead.

In this paper, I first discuss a couple of possible but incomplete definitions of software productivity and list some of the attributes on which it depends. Also included are different viewpoints about software productivity and associated measurements. There are many other aspects of software productivity like advantages, means to improve, etc., which one may discuss, but I intentionally omit them here. However, I do discuss various issues related to software productivity management.

DEFINITION AND DIFFERENT VIEWPOINTS

It should again be emphasized at this point that this paper will discuss software productivity as opposed to programmer productivity. Thus, the definitions and their corresponding metrics that we refer to are not intended as a means of setting goals for the evaluation of individual performance.

This section discusses a couple of definitions of software productivity and some dependent attributes. It also discusses an economic viewpoint, a management viewpoint, and a way to look at software productivity as the software progresses through various life-cycle phases.

Traditionally, productivity has been defined as follows.

$$\text{Productivity} = \frac{\text{output}}{\text{input}}$$

This is simply an efficiency ratio. Depending upon the type of organization and its primary task, the output could be measured in terms of physical units, the dollar amount (after accounting for inflation, i.e., in terms of constant dollars), equivalent man-hours (for service organization), etc.; the input could be measured in terms of units consumed, dollar amount consumed, etc. For example, in a manufacturing environment, the outputs are hardware products, and the profits obtained by their sales; the inputs are the cost of the material used, energy, and labor.

One of the major difficulties faced in defining software productivity is that we do not know what constitutes the output and what constitutes the input. Some of the output attributes on which productivity depends are as follows.

1. Software quality
2. Software reliability
3. Software usability
4. Software maintainability—this may depend on the methodology used to develop software, and on many other factors
5. Software verifiability
6. Timely delivery
7. Customer satisfaction

Some of the input attributes on which productivity depends are as follows.

1. Personnel cost to develop software
2. Cost of equipment
3. Capital investment
4. Incentives given to software personnel

These lists are clearly not exhaustive.

There is always an economic viewpoint when looking at productivity. The economic definition of productivity along with the associated input and output factors for each term is shown as follows.

$$\begin{array}{ccccc} \text{Output value} & & \text{Quantity used} & & \text{Initial price} \\ \uparrow & & \uparrow & & \uparrow \\ \text{Productivity} & = & \text{Product} & \times & \text{Price recovery} \\ \uparrow & & \uparrow & & \uparrow \\ \text{Input value} & & \text{Quantity produced} & & \text{Initial cost} \end{array}$$

Along the same lines, the March 1981 IEEE Software Engineering Productivity Workshop chose to use an economic definition of software productivity for their deliberations.[4] Their definition is as follows.

$$\text{Software productivity} = \frac{\text{values}}{\text{costs}}$$

where *value* is defined in profit-oriented terms either as an increase in revenues attributable to the product, or a cost reduction resulting from the use of the product. This definition focuses attention on management decisions rather than on programmer output. Table 1 lists the fundamental elements of economic measurement.

Another way of viewing software productivity is to concentrate on the software system not only in its entirety, but as it progresses through various phases in its life-cycle. The IEEE Standard[5] defines software life-cycle as the period of time that starts when a software product is conceived and ends when the product is no longer available for use. Typically, it includes a requirements phase, design phase, implementation phase, test phase, installation and check-out phase, operation and maintenance phase, and sometimes a retirement phase.

Software productivity of any software development process as a whole depends upon the productivity as obtained during its progress through various life-cycle phases. The output of one life-cycle phase becomes input for the subsequent phase; thus, any metrics used to measure software productivity must predict the resources required for the subsequent phases of the life-cycle based upon the characteristics of the software at a given point in its life-cycle.

TABLE 1—Fundamental elements of economic measurements (from Munson, J. B., and Yeh, R. T., Report by the IEEE at the Software Engineering Productivity Workshop, San Diego, March 1981).

	Static	Dynamic
Costs	Cash	Rate of spending
	Human effort	Present value of money
	Mechanical effort	
Values	Financial	Present value of revenue market penetration.
	Informational	
	Quality of Life	Present value of information.

During the lifetime of a software system, it is not very uncommon to find that a life-cycle phase is iterated many times. Any measure for software productivity should be inversely proportional to the number of times a phase is iterated in the lifetime of a system because the iterations of most life-cycle phases induce iterations of some other phases thereby driving up the development cost.

It should be born in mind that productivity measurement result takes on meaning only when compared with other results of productivity measurements. Things which a productivity measurement may be compared with include the following.

1. Actual historical data
2. Existing standards in the industry
3. Other currently available actual data

Changes in future productivity metrics depend on the extent to which industry participates in using standard metrics. (We do not have such standard metrics yet, but IEEE has set up a working group on standards for software productivity metrics, and we may expect their recommendations by the middle of 1986.) Greater industry participation implies better understanding of the metrics. This in turn implies improvements in its definition which further provides more accurate measurements.

Management's viewpoint of software productivity is a little different because they have a different set of priorities and responsibilities. Certainly, they like the economic definition of productivity, but they view software productivity as the ability of a corporation to develop software. A metric for software productivity is useful to them if the measurements help each level of management to make appropriate decisions with regard to software development while keeping the economic goals of the company in mind. This means that for each level of management, productivity takes on a different meaning depending on the concerns of that level of management. The industry-wide concern may be technological capability and international competition; corporate level concern may be profits and market position; unit management's concern may be resource allocation; project management may worry about actual progress against designated milestones; the project team's concern may be the integration of individual products; and an individual's concern may be the quality of the product and the work rate.

The variety of viewpoints and the fundamental difficulty of measuring software make it extremely difficult to establish a standard definition of software productivity. It is interesting to note that the IEEE Standard Glossary of Software Engineering Terminology[5] defines many attributes of software including *quality* and *reliability,* but does not define *software productivity.*

SOFTWARE PRODUCTIVITY MANAGEMENT

Software productivity management is quite different from productivity management for a hardware manufacturing unit. This is true because

1. Software attributes that form software productivity issues are not all clearly understood, and
2. No appropriate metrics for measuring software productivity exist.

As if this were not enough, the problem of software productivity management is compounded by the fact that there is a shortage of computer scientists skilled in managerial issues. This is due to the fact that computer science is a relatively new area of expertise. In software houses, first line managers are usually chosen for their technical skills only, whereas in most cases the higher-level managers do not understand the software development process completely. The best way to alleviate this problem is to train more computer scientists to be managers.

There are many issues which can be considered central to software productivity management. With a little help from Munson and Yeh,[4] I list some of them.

1. The role of software in the organization
2. The quality of software management
3. Understanding the approach to software development
4. How management fosters new ideas and creativity
5. The use of proper personnel for available tasks
6. The availability of software development tools

I will not discuss each of these issues in detail directly, but ask that you consider a productivity management process that concentrates on most of them.

A productivity management process consists of the following two fundamental issues discussed in detail below.

1. Efficient use of the following resources
   a. manpower
   b. materials
   c. technology
2. Satisfaction of consumer requirements related to the following
   a. cost
   b. schedule
   c. performance

Manpower

Management must use available manpower efficiently. To achieve this, it must identify individuals with special skills and assign them to projects so as to maximally exploit those skills. To get the most out of people, it is necessary to allow them direct access to top-level management, opportunity for outside recognition, personal professional growth, and increased benefits. A dual ladder with salary, status, and visibility for the most skilled software developers is required.

Software managers must also assume responsibility for a smooth transition from one phase of the life-cycle in the software development process to another. Frequently, various life-cycle phases overlap. Even so, the transition from a partially complete phase to a new one should be smooth. This is particularly important for transition from the implementation to the testing and maintenance phases where about 50 to 60% of software cost is incurred. Thus, software managers should make sure that, after implementation, there is an orderly transition to the maintenance phase, particularly with respect to the assignment of personnel already familiar with the project to subsequent phases. This maintains continuity during the most important transition in the software development process, thereby reducing the total system cost.

Material

Management needs to provide materials to software developers so that productivity can be consistently improved. These materials could be provided in many forms including increasing travel budgets for personnel to attend professional seminars and conferences that keep them abreast of the latest developments in the field; increasing the contents of the library to make sure that the latest research results are available to the software staff in time; improving office facilities; and providing more terminals and better secretarial support to the software engineers.

Technology

Management should provide the software staff with appropriate equipment and the means to use it such that they do not lag behind their competition. Management must realize that such tools are very important instruments to achieve productivity gains. It is important to note that hardware costs in the computing industry have gone down consistently over the last 15 to 20 years whereas software costs have continually increased. Management should recognize this and be prepared to buy the hardware needed to simplify the tasks of its software staff. Providing the latest technological developments in the software area to the software engineers can also increase productivity.

Cost Management

Management must keep the cost of software development and maintenance as low as possible. This provides the organization with a competitive edge and increased profit margin on the product. There are many ways of managing and controlling the cost of software development.

Schedule

Management at all levels must assure that software development is taking place within the appropriate budget and on schedule as the software progresses through its life-cycle phases. Project managers play a big role in maintaining and adhering to prescribed schedules.

Performance

Management must guarantee the performance of the delivered software at or above the required levels of customer satisfaction. Software productivity metrics are directly proportional to the performance level of software and customer satisfaction because both increase the value of the software. Additionally, as software productivity depends on software quality, it is the duty of managers at different levels to ascertain that at the end of each phase of the life-cycle, the developed software meets its requirements specifications. Verification and validation of software with respect to its specifications are important aspects that ensure good software performance in the future.

ADVANTAGES OF SOFTWARE PRODUCTIVITY MANAGEMENT

> Management actions and reactions to the increasingly pervasive impact of software development on corporate products and processes have considerable impact upon a corporation's ability to develop software, i.e., its productivity. Each level of management has a natural set of decisions to make which impact productivity.[4]

When some results of productivity measurement become available to managers at all levels in the hierarchy, the decision making capability of management increases. Once the relationship of software to corporate products and processes is recognized, the productivity measure becomes the basis for evaluation of capitalization for software development, corporate strategy for acquiring additional technology, and evaluation of the performance of groups within the company and other organizations. Such measures also help to identify the specialized skills required to improve productivity and thereby support corporate goals.

Middle-level management can use productivity measures to explain to the top-level management how corporate goals can be achieved through software development. Measurements of productivity allow for the tracking of progress within budget and schedule. Sometimes, they can be useful in imposing a methodology on the development or maintenance process.

For software managers, productivity measures allow for the accurate prediction of group performance. This allows management to accurately project the cost and schedule of subsequent projects. At the upper levels of management, it may be used to decide on the life or death of a project or to evaluate the benefits of new tools, etc. Appropriate measures may also be used by managers to identify support requirements and to foster creativity, initiative, and innovation.

SUMMARY

The state of the art of software productivity, its associated measures, and its management is that it is still an art, not a science. We know quite a bit about the attributes that determine the cost of software production processes or "input" to the productivity measurement, but we have yet to identify many software attributes that determine the value of a software system. This, to my mind, is the biggest stumbling block in precisely defining software productivity. Lack of a precise definition results in our inability to understand software productivity and to measure it effectively.

It is clear that the study and management of productivity requires a measurement discipline. Measurement is as fundamental to programming as it is to any other area of engineering. If programming is to keep pace with technological advances in other areas of engineering, productivity improvement is critical. But, improvements in measurement techniques have not kept pace with the technological improvements in software engineering. Unless this happens, software costs will be ever increasing.

Software managers at all levels have to play a crucial role in these productivity improvements. It is not going to be an easy task, but the economic realities of the present day environment are forcing us to analyze and improve our software productivity management. We will have to look at traditional management techniques and improve and/or adjust them so that they can be applied to the software development process.

REFERENCES

1. Boehm, B. W. *Software Engineering Economics.* Englewood Cliffs: Prentice-Hall, Inc., 1981.
2. Steel, T. B. Jr. "A Note on Future Trends." in P. S. Nyborg (ed.), *Information Processing in the United States: A Quantitative Summary.* Montvale, N.J.: AFIPS, 1977.
3. Boehm, B. W., Brown, J. R., Kasper, H., Lipon, M., MacLeod, G. J., and Merritt, M. J. *Characteristics of Software Quality.* Amsterdam: North-Holland, 1978.
4. Munson, J. B., and Yeh, R. T. *Report by the IEEE Software Engineering Productivity Workshop.* San Diego, March 1981.
5. IEEE Standard Glossary of Software Engineering Terminology, *IEEE Standard 729-1983.*

Software maintenance: a different view

by NED CHAPIN
InfoSci Inc.
Menlo Park, California

ABSTRACT

The problems identified by those supervisors and managers of application software maintenance who are closely involved in this type of work are shown in this paper to form into sets of patterns, with ties between them. The implications of these sets of patterns and associated ties differ in several important ways from what has been previously reported in the field. An analysis of the responses obtained from a major survey gives a basis for identifying the sets of patterns. Four interpretations of the analysis results integrate the findings with prior work reported in the field. The conclusions offer three significant insights potentially useful to management in the area of application software maintenance.

INTRODUCTION

Part of the folklore of the software side of the computer field is that managing and supervising software maintenance work is fraught with problems. Some significant efforts have been made to identify these problems on an experiential basis (such as the work included in the volume edited by Parikh[1]), on a research basis (such as the work done by Lehman[2]), and on a survey basis (such as the work done by Lientz and Swanson[3]). Such efforts have yielded a multitude of ideas, but not much that integrates them, especially as a basis for action. The work reported in this paper provides both new data and a more coherent integration than previously available.

The term *software maintenance* is used in this paper to refer to the combination of four main kinds of activities done with existing application programs and systems. Another term sometimes used for this combination is *application software support*. One kind of activity included in the combination is making enhancements or modification to the capability of the existing software to meet the request of users. Usually this arises from the users' changed views of their requirements as, for example, when a user wants to add a new profit center to reflect the production and marketing of a new product. A second kind of activity included in the combination is making adaptations in the existing software to work in a changed data processing environment. An example of this is the altering of existing programs and systems to make use of the features of newly-introduced communications systems software. A third such kind of activity is the making of corrections and routine form changes in the existing software to preserve its current functionality. Two examples of this are the return to a running state of a program which has abnormally terminated during execution, and the change of spelling of a data name to make it consistent among all programs within a system. The fourth such kind of activity is in responding to user needs in the use of the program or system as, for example, when the user asks why the computer is producing totals on one report which do not reconcile with the totals shown on another report produced by the computer as an output from the same system. These definitions differ in detail from some others in the field,[4] but they preserve a useful degree of congruence with the most popular ones[3] while correcting for some of their possible oversights.

PURPOSE

The major survey work reporting on the problems facing managers and supervisors of software maintenance was done by Lientz and Swanson.[3] This work used a questionnaire listing 26 "problems in maintaining the application system." The respondent was to rate each problem for five categories of concern ranging from "no problem at all" to "major problem." Respondents, who were the main administrative officers within their data processing organization, often delegated the actual completion of the form to a subordinate and then reviewed the responses.

Two criticisms of this methodology focus on the position of respondents and on the means of acquisition. Respondents rarely were the personnel closest to the problems (i.e., first line supervisors and their immediate superiors). Thus, the view obtained was from too high and distant a vantage point. The means of acquisition allowed no opportunity for respondents to list the problems they were experiencing, but asked them to rate from a set list of problems; respondents could not add to the list problems that may have been far more significant to them than those items preselected for them. Respondents also did not have the opportunity to restate the problems in a form that reflected the realities of their own situations and, thereby, change the evaluations they recorded.

Survey work by the National Bureau of Standards on software maintenance problems[4] although not reporting numeric values, reports a list of problems having little in common with the Lientz and Swanson list. While it may be argued that the Federal Government environment is different from that of the private sector, other Federal survey work has indicated that the differences are minor for software maintenance.[5] Hence, the difference between the two lists is probably significant.

An IBM survey[6] of 25 data processing installations reporting on major maintenance problem areas gave a list more like that of the National Bureau of Standards than that of Lientz and Swanson, but it assigned no numeric values for either relative frequencies or importance.

From the study of these efforts and those of others such as Dean and McCune,[8] it appeared that we had yet to find out what the supervisory and management personnel closest to the actual software maintenance work see as the problems. With such knowledge, higher level managers of application software maintenance would be in a better position to select appropriate action to alleviate the problems. Hence, the purpose of the study reported here was to survey what the supervisory and management personnel closest to the software maintenance work see as the problems, and then, if possible, analyze and interpret the gathered data.

SURVEY

Responses to 260 questionnaires were obtained from personnel employed at 123 different data processing installations.

Responses were solicited from personnel who supervised others in the performance of application software maintenance work. Thus, job title was not a criteria for solicitation. The job titles most commonly encountered were supervisor, project leader, lead analyst, lead programmer, programming manager, and data processing manager. At only one installation were more than half of the eligible personnel solicited; higher level supervisors or managers generally designated the personnel to respond to the questionnaire. The data were gathered over more than a year's time from installations scattered all across the United States, but with an emphasis on the Eastern seaboard. Two Canadian installations were also surveyed. While the distribution of responding organizations covered the expected range of industries, manufacturing, particularly electrical goods manufacturing, was under-represented. Government installations were over-represented (contributing just over 29% of the number of installations).

To meet the objective previously identified, a very brief and open-ended recall response was solicited in the following form:

> Our major problems in maintaining programs are . . .

followed by space for three responses.

The respondents provided 769 responses for an average of 2.96 responses per questionnaire and a range of responses from zero through seven per questionnaire. From some installations, personnel responding held similar views and shared the task of filling out a questionnaire; from others, views varied widely. Since the respondents selected their own wording to express their responses, great variation resulted ranging from system characteristics to coding tricks to management style to organizational policy. This presented a challenge in the subsequent classification of responses. However, an independently-done 100% check of the assignments made resulted in questioning less than two percent of the classifications assigned to the responses. The tabulation of the responses as classified is summarized in descending order of frequency in Table I.

In interpreting the frequencies of the responses as classified, some general background may be helpful. Since the question asked for major problems and provided space for more than one response, the interpretation should be different from what it would have been if only one response had been allowed—a wider spread and more variation are to be expected. Without a limiting preset list, the variations in response expression can be expected to reflect differences in perceptions; this is especially true since no direct or indirect "mental set" was provided through verbal or written communication prior to the respondents' completion of the questionnaire.

The classifications of responses with the highest degree of coherence within the classes are documentation, personnel shortage, and lack of time. Examples of responses in these classes are, respectively, "Poor documentation," "Inadequate staffing," and "Not enough time." The classes with the weakest degree of coherence are Miscellaneous, Maintenance administration, and Lack of planning. Examples of two responses from each of the latter two classes are "Coordinating changes" and "Making reports," and "Conflicting user requests" and "Not anticipating down-time," respectively. The three responses in Miscellaneous are "Productivity," "Difficulty of user requests," and "Retrofit." The context provided by the other statements was used to help decide how to classify particular responses when alternatives seemed attractive and reasonable.

TABLE I—Tabulation of survey responses by frequency

Classification	Frequency
Documentation	134
Personnel capability	80
Personnel shortage	42
Software complexity	38
Communication with users	34
Software not well structured	32
Interactions and interfaces	30
Poor techniques used	27
Lack of time	25
Data access and definition	23
Program characteristics	22
Inadequate resources	22
Estimates, schedules, and priorities	21
Debugging and testing	20
Personnel turnover	20
Old code	17
Poorly designed for maintainability	16
User knowledge	16
User involvement in maintenance	13
Many languages	13
Patched patches	13
Change control procedures	12
Software not written here	12
Volume of work	12
Lack of planning	12
Maintenance administration	10
Standards	9
Big programs and systems	7
Morale and motivation	7
Multiple versions	5
Mandated changes	5
Meet user's new needs	4
Communication with others	4
Cost	4
Backlog	3
Keeping quality up	2
Miscellaneous	3
Total responses	769

ANALYSIS

After consistency and reliability checks, the classified responses were grouped at a higher level. This higher-level grouping is shown in Table II along with the approximate percentage allocation of the responses and summarized in Figure 1.

A closer look at the grouping of these responses is helpful. Two concerns dominate: (1) software characteristics, and (2)

TABLE II—Groupings of the classifications shown in Table I

Group and Classification	Frequency
Software characteristics (48%)	372
SD—Documentation	134
SC—Software complexity	38
SS—Not well structured	32
SI—Interactions and interfaces	30
ST—Poor techniques used	27
SA—Data access and definitions	23
SP—Program characteristics	22
SO—Old code	17
SM—Poorly designed for maintainability	16
SL—Many languages	13
SH—Patched patches	13
SB—Big programs and systems	7
Personnel factors (20%)	149
PC—Personnel capability	80
PS—Personnel shortage	42
PT—Turnover	20
PM—Motivation and morale	7
Maintenance management (9%)	70
ME—Estimates, schedules, and priorities	21
MC—Change control procedures	12
MP—Lack of planning	12
MM—Maintenance administration	10
MS—Standards	9
MX—Cost	4
MQ—Keeping quality up	2
Environmental factors (8%)	64
ET—Lack of time	25
ER—Inadequate resources	22
EV—Volume of work	12
EM—Mandated changes	5
Activities involving software (8%)	57
AU—Communication with users	33
AD—Debugging and testing	20
AO—Communication with others	4
User relations (5%)	37
UK—User knowledge	16
UI—User involvement	14
UN—Meeting users' new needs	4
UB—Backlog	3
Distribution of software (2%)	17
DA—Software not written here	12
DV—Multiple versions	5
Miscellaneous (0%)	3
CM—Miscellaneous	3
Total responses grouped (100%)	769

personnel factors. Software characteristics reflect the character of software that the personnel work on (application programs and systems, for the most part). The importance of some of these characteristics for maintenance has been noted elsewhere.[7] The personnel factors reflect the character of the personnel who are assigned to work on this software and who must deal with its characteristics. Those who responded to the survey see software maintenance problems as arising mostly from the form of the application software and from the personnel resources they can assign to do the work.

Some respondents saw these problems in detail, identifying specific characteristics of the software and personnel as contributing to the problems. Others identified more general and abstract characteristics. However, their concerns were, in effect, still much the same—trying to match the characteristics of software with appropriate personnel. Making that match work in practice is a common and critical part of the job of the supervisor and manager of application software maintenance.

To a much lesser extent, respondents identified five other problem sources contributing a total of about one-third of the problems. The largest group of these is maintenance management. These are problems arising from the action or inaction of management personnel outside of the user areas as, for example, in planning, coordinating, directing, controlling, and evaluating data processing application software maintenance, and in securing and applying resources for its accomplishment. Some of these concerns involve management interaction with users as, for example, in setting priorities and schedules.

Two other groups of problems are seen as closely related to the management group; one appears to arise from the organizational environment in which application maintenance work is done, including such demands as speed of response; the other is the set of activities which involves application software. Two such activities receiving overwhelming attention in this survey are communicating (especially with users) about maintenance work on the application software, and debugging and testing the software as a part of the maintenance work. In both cases, the software is seen as the subject of activities to improve the service it can render to the user.

User relations and distribution of software in the organization largely complete the list of problems. These accounted for less than 10% of the problems (user relations being more prevalent than software distribution).

It is interesting to note that a survey of 12 computer sites in Brazil came up with a list of nine most-frequent problems similar to that reported here.[9] While the Brazilian sample was too small to permit quantification of the sort reported here, the importance-ranking matched quite closely to that shown in Table I.

INTERPRETATION

The most significant interpretation of the survey results is the surprisingly small emphasis placed on users requests as a source of problems. While it is widely held that application software maintenance is a service rendered to the user, and that keeping the user satisfied is the major task,[10] only one of the 769 responses saw user demand as a major problem. This ratio indicates that satisfying work requests, whether for corrective or enhansive maintenance, is rarely viewed as a major problem. Providing modified functionality in the software also gives rise to relatively few problems. Thus, altering the performance of the application software is also not seen as a major problem.

What, then, are the major problems? With few exceptions, they are seen to be the *circumstances* or *conditions* under which the altering of the performance of the software is to be done. Here, respondents had a litany of complaints about the

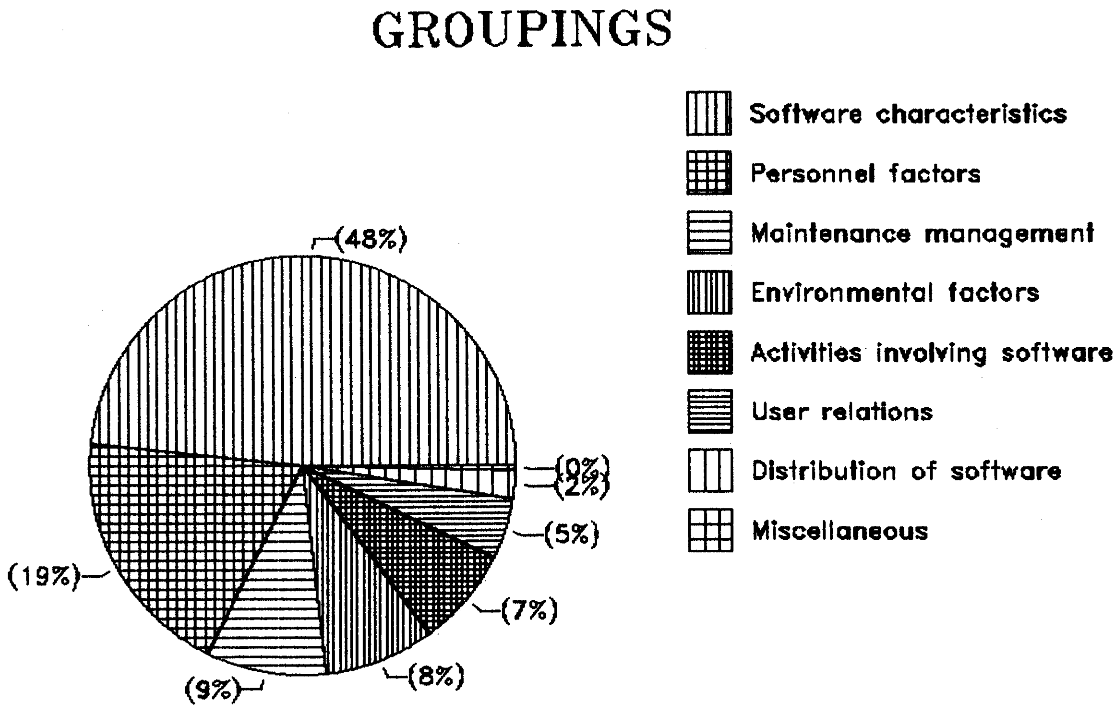


Figure 1—Summary of the groupings of the classifications shown in Table I

adequacy of documentation, time allowed by users to accomplish tasks, poor source code structure, complex systems or programs, shortages of personnel who know the language, shortages of terminals, etc. It is as if maintenance personnel were being asked to do ballet on a muddy dirt road while dodging traffic in the sleet—the task can be done, but the circumstances grossly hinder good performance and actually increasing the amount and difficulty of the work to be done.

A second significant interpretation of the survey results is the absence of perceived problems in improving personnel productivity. Productivity as a problem was noted only once out of 769 responses, but many of the problems mentioned would be relevant to productivity if the respondents were to see it that way. Examples of this appear in the environmental group (lack of time, inadequate resources, and volume of work), the personnel factors group (personnel capability, personnel shortage, and turnover), the management group (estimates, schedules, and priorities, and lack of planning), and the user relations group (backlog).

The respondents paint a picture of more work to do than can be done in the time available with the resources committed. Since the respondents indicate that they are largely blocked from acquiring additional personnel resources, one would expect them to mention problems reflecting their efforts to get more work done with existing personnel and to get support from the use of other resources (for example, tools and methods such as code generators, prototyping, and software engineering). But the respondents did not perceive things this way. Instead, the bulk of the problems described are of the "litany" type noted above. Where one would expect to find descriptions such as "Can't get rapid prototyping introduced," or "Walkthroughs fail to find more bugs," the respondents instead mentioned problems such as "Programs being worked on are not well-structured," and "Hard to keep personnel who know the system."

Supervisory and management perception of problems is shown to be reactive, largely ignoring productivity, rather than pro-active, applying productivity enhancements. This interpretation is consistent with the findings on management style in application software support reported at the Software Maintenance Conference in 1984.[11,12]

Managers in application software maintenance are in a situation (by their own tacit assessment) calling for sharply increased personnel productivity, yet they are not reporting any problems in increasing productivity. According to higher level managers and users,[1,3,13] management in application software maintenance is not making significant progress at increasing personnel productivity. This may, in part, account for the lack of productivity problem perceptions among respondents to the survey.

A third significant interpretation of the survey results is in the area of software characteristics. In that grouping, documentation predominated, being mentioned nearly twice as frequently as any other classification. This stress on documentation is consistent with a report showing that improving documentation is a significant route to reducing the cost of application software support,[14] and with the widely held view that maintainability must be designed in.[15] Furthermore, when documentation is combined with other classes in the software characteristics group, the collective results offer firm support to other findings reported on management perceptions of these software characteristics.[16] While these results indicate that software characteristics are the most common cause for concern, these problems are, by and large, creations of current and past policies and practices within the respondents' own organizations.[16] It is noteworthy that the concern with priorities and planning mentioned in the maintenance management classification does not include any mention of problems in changing these policies and practices. The policies and practices are still going on largely unchanged.

This "dog chasing its tail" situation will not change by itself, but a glimmer of hope for change is offered through four responses describing some problems in meeting users' new needs. Some of these also bode of even worse conditions to come, as in "downloading data for microcomputers." As package use extends, this picture may change for the worse as more "alien" software finds its way into the organization. As organizations acquire and attempt to run the same programs and systems on computers of different makes and sizes, this picture will likely deteriorate further. Some of these changes will probably then appear as additional problems in the user relations area.

A fourth significant intepretation of the survey results is in the perceptions by the respondents of users as sources of problems. Personnel at different levels in the organization see software maintenance differently,[17] but at the level surveyed, they see users as sources of problems (primarily based on the inadequacy of specifications or service requests). This is viewed in terms of communication difficulties; not in terms of what is asked. Although prior studies have often not quantified the importance of users as sources of problems, the number of major problems from this source, as seen in this survey, is much lower (less than 3%) than that implicitly reported elsewhere (for example, by Martin and Osborne[4]). Yet at the same time this survey offers support for the finding by Lientz and Swanson[3] of the significance of the users' knowledge of the software, for it is perceived as a problem by the supervisory and management levels responding to this survey.

CONCLUSIONS

Integrating the results of this survey with other work in the field leads to several conclusions.

1. Higher level management attempts to improve application software maintenance will be fruitless until they address what those closest to the situation perceive to be the dominant problems—software characteristics and personnel factors. As reported elsewhere,[10] these are not commonly held management objectives, but the finding here offers support for the direction suggested by Martin and Osborne.[4]
2. Those closest to the situation do not perceive themselves and their subordinates to be active in productivity improvement.
3. Because the software characteristics and environmental factors are but slowly tractable, probably the most rewarding areas in which to apply effort to improve application software maintenance in the short term are personnel factors (most importantly), and (to a lesser extent) the maintenance management process and the activities involving software.

REFERENCES

1. Parikh, Girish, Editor. *Techniques of Program and System Maintenance.* Lincoln: Ethnotech, Inc., 1980.
2. Lehman, Meir M. "Programs, Life Cycles, and the Laws of Software Evolution." *Proceedings of the IEEE.* (Vol. 68, 9), September 1980, pp. 1060–1076.
3. Lientz, Bennet P., and E. B. Swanson. *Software Maintenance Management.* Reading: Addison-Wesley Publishing Co., 1980.
4. Martin, Roger J., and W. M. Osborne. *Guidance on Software Maintenance.* Washington: National Bureau of Standards, 1983.
5. Comptroller General. *Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged.* Washington: U.S. General Accounting Office, 1981.
6. Fjeldstad, R. K., and W. T. Hamlen. "Application Program Maintenance Study." in Parikh and Zvegintzov (Eds.), *Tutorial on Software Maintenance.* Los Angeles: IEEE Computer Society, 1983, pp. 13–27.
7. Chapin, Ned. "Attacking Why Maintenance is Costly—A Software Engineering Insight." *Proceedings of the IEEE Software Maintenance Workshop.* Los Angeles: 1984, pp. 251–252.
8. Dean, Jeffrey S., and Brian P. McCune. "An Informal Study of Software Maintenance Problems." *Proceedings of the IEEE Software Maintenance Workshop.* Los Angeles: 1984, pp. 137–139.
9. Valdesuso, Carlos. "Manutenção: porã de navio?" *SCI Resumos Téchnicos,* (Rio de Janeiro), September 1984.
10. Chapin, Ned. "Software Maintenance Objectives," *AFIPS, Proceedings of the National Computer Conference* (Vol. 52), 1983, pp. 779–784.
11. Chapin, Ned. "Productivity in Software Maintenance." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 349–352.
12. Osborne, Wilma M. "Life Cycle Requirements of Software Maintenance." in *EDP Software Maintenance.* Silver Spring, Md.: U.S. Professional Development Institute, 1984, pp. 430–478.
13. Richardson, Gary L., and C. W. Butler. "Organizational Issues of Effective Maintenance Management." *AFIPS, Proceedings of the National Computer Conference* (Vol. 52), 1983, pp. 155–161.
14. Guimaraes, Tor. "Managing Application Program Maintenance Expenditures." *Communications of the ACM,* 26, 10 (Oct. 1983), pp. 739–746.
15. McKee, James R. "Maintenance as a Function of Design." *AFIPS, Proceedings of the National Computer Conference* (Vol. 53), 1984, pp. 187–193.
16. Chapin, Ned. "The Profile of Success in Managing Support Services." in *EDP Software Maintenance.* Silver Spring: U.S. Professional Development Institute, 1984, pp. 194–201.
17. Colter, Mel A., and J. D. Couger. "Management and Employee Perceptions of the Maintenance Activity." *Proceedings of the IEEE Software Maintenance Workshop.* Los Angeles: 1984, p. 130.

The impact of implementing a rapid prototype on system maintenance

by JOHN L. CONNELL
Martin Marietta Denver Aerospace
Denver, Colorado
and
LINDA BRICE
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

Rapid prototyping is a popular new way of developing software applications in both business and scientific environments. The technique is invaluable in providing communication between customers and developers; but like any other methodology, it can fail if not used properly. This paper compares and contrasts two case studies of prototyping endeavors. One appears to be a success, and the other had major problems and may result in total failure. The reasons for the phenomena are probed, concluding with an observed successful prototyping life cycle, which, if abandoned, may negate the positive aspects of the entire prototyping process. The necessary phases of rapid analysis, database development, menu development, function development, prototype iteration, detailed design, and implementation have been well documented. It is strongly suggested that three new phases—planning, modeling, and training—be added to the cycle for a better chance of successful rapid prototyping.

IMPLEMENTING THE PROTOTYPE: TWO CASE STUDIES

Much attention has been devoted to the techniques of rapid prototyping in recent literature. Some organizations have been convinced that the idea is viable and have begun to use it as a development tool. Often, when rapid prototyping is promoted, the promotion is accompanied by some caveats, such as "be prepared to throw the prototype away" and "do not attempt to implement a prototype; detailed design will still need to be carefully prepared." This paper describes two development projects in which rapid prototyping was employed. One succeeded and one failed. In both cases the standard caveats were specified, and during both projects some surprises occurred. The psychology of some of the situations was that the cautions were intellectually considered, but not internalized or fully accepted. From the experience gained during the development of these two prototypes, some of the caveats have become recognized as simply too important to be ignored under any circumstances. Ignoring them has clearly become a sure way to make the implementation of a prototype fail.

The following material illustrates the critical differences in the techniques used to develop and implement each of the systems. The lessons learned from this comparison are then generalized into guidelines for success and absolutes for avoiding failure in rapid prototyping. One conclusion that must be emphasized at the beginning: It may often be the case that the rapid prototype itself is done correctly, but the surrounding life cycle phases are not, and that this is the primary cause of failure, not the techniques of rapid prototyping. There is a danger that too many failures in the application of a new technique will cause critics to rise up in force and insist on its abolition.

For both systems INGRES, a relational database management system, was used to develop a rapid prototype according to methods described in the literature. The authors observed prototype development in separate organizations. In both cases the systems were implemented by persons who were not closely involved in prototype development. Both prototypes were good ones in that they were working models that were developed rapidly and met with user approval. One became a usable system, but the other may be abandoned, for reasons that will be described.

The Resource Control System Prototype: a Failure

This prototype was developed at Company A with the intention of quickly replacing an existing system, which, it had been determined, did not adequately meet user requirements and was difficult to modify. The purpose of the application was to capture data on the cost of development, modification, and operation for all software application systems. This information would then be reported to system developers, project managers, organization management, and end users. The system was to be interactive and allow for accurate forecasting of effort and comparisons of forecasts with actual costs. The primary users for this system were the software developers themselves.

Software developers at Company A are expected to fill out time sheets weekly, showing how many hours were worked on various system phases. These data, along with hardware operation costs, are the primary input of cost information into the system. Monthly forecasts of these costs are also required and input into the system. The system must then produce a large variety of reports: forecast effort, weekly timesheets, costs by system, costs by user, and many more. These reports have multiple uses, such as forecasting tools, customer billing reports, and hiring justifications. Figure 1, a dataflow diagram, gives a high-level view of the inputs and outputs for the resource control system.

The useful purposes for information from this system are almost unlimited. Unfortunately, the old system did not provide this information in an acceptably accurate, timely, or

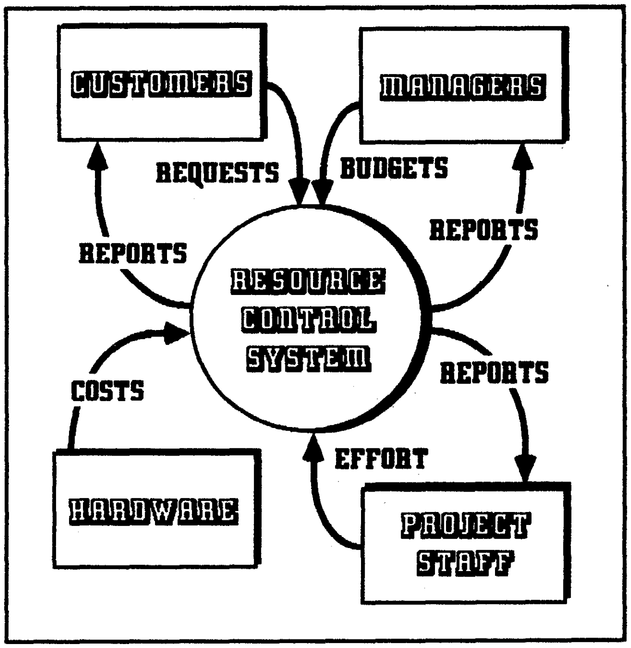


Figure 1

user-friendly fashion. Therefore it was decided to use rapid-prototyping techniques to model possible resource-efficient solutions. A new system was to be brought up quickly on a low budget. This project could be considered a failure, since the old system may have to be continued and the prototype scrapped.

The Comprehensive System Development Environment Prototype: a Success

This prototype was developed at Company B with the intention of helping to define user requirements for a set of new automated software development tools. It is a brand new application in the sense that an automated system performing the same functions did not exist previously. Its purpose was to provide, through the use of a common project database, a set of tools to help analysts, designers, programmers, project managers, and other project support personnel with the preparation of software system specifications and documentation. Figure 2 is a visual representation of the comprehensive system development environment. Creation of the total system will be a very elaborate project, originally estimated to require about 50 person-years of development effort.

Management decided to use rapid prototyping to develop an interim product: a requirements document generator. In this way it was felt that, through demonstrations of the system to potential users, design details of the common project database and the user interface could be worked out most efficiently. The requirements document generator is based on the assumption that a requirements analyst will be using structured analysis incorporating dataflow diagrams, process descriptions, and a data dictionary. Further, it is assumed that the printed requirements document will be in military standard format. The prototype was developed to allow information generated during structured analysis to be stored in the common project database in a way that would allow these data to be combined with standard boilerplate text and formatted correctly into a requirements document that would meet with customer approval. Figure 3 is a high-level dataflow diagram showing how the requirements generator interfaces with other parts of the comprehensive system development environment.

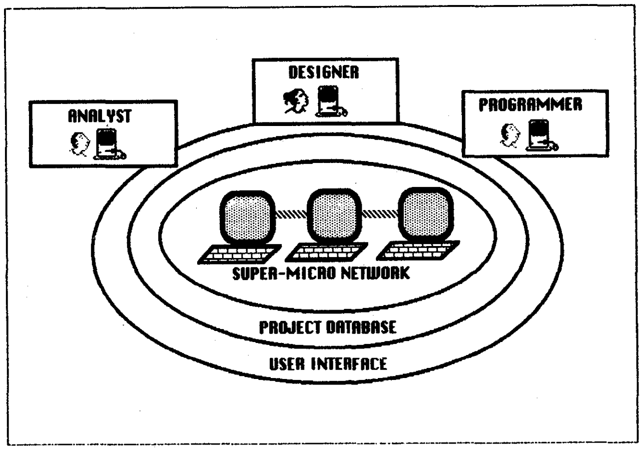


Figure 2

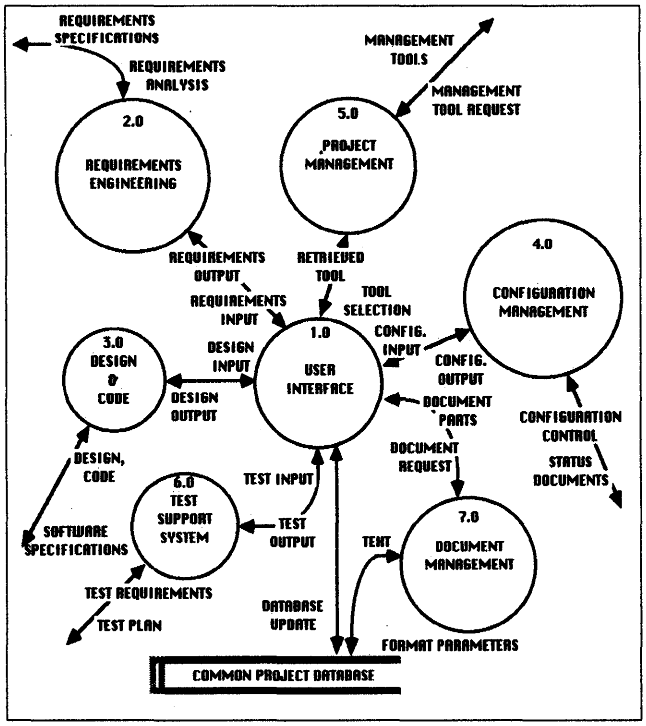


Figure 3

One of the problems frequently encountered in using structured analysis correctly is the amount of manual effort involved in checking for errors or violations of structured-analysis guidelines. Therefore, consistency-checking procedures were built into this prototype. These procedures use the features of a relational database to compare the analysis information entered against a set of guidelines and output a set of reports listing the violations. The prototype then became a tool to assist analysts with structured analysis, rather than simply a document formatter and generator. As a test of the workability of the system, the requirements for the comprehensive system development environment itself were entered into the prototype, and a requirements document was generated in military standard format.

It was found that the resulting requirements document was prepared in an unusually short period of time, with much higher than usual accuracy, because of the consistency-checking features of the prototype. Problems encountered during preparation of the document caused refinements to be made to the prototype, resulting in a more robust system. Finally it was discovered that some of the functional modules developed for the requirements document generator could be used with slight modifications in the development of other system functions, such as the design document generator. This project succeeded, yet prototyping was carried out in much the same fashion as for the failure cited above. There were some setbacks, but they were not due to prototyping. They probably would have happened anyway, and because the project was developed as a prototype, they were more easily corrected. The differences in project management between the two case studies will be discussed to show the reason for the difference in outcome.

THE (REVISED) LIFE CYCLE OF A RAPID PROTOTYPE

The authors have published prototyping life cycle phases in previous works.[1,2] On the basis of recent experience gained from the projects just described, new phasing elements are believed to be necessary in the prototyping life cycle. The previous version of the life cycle called for a project to begin with a phase called *rapid analysis,* in which preliminary user interviews were to be used to develop an intentionally incomplete, high-level, paper model of the system to be prototyped. A first-cut prototype would then be created with a relational database structure, developing menus and functions using fourth-generation techniques to produce a working model. Next a phase called *prototype iteration* is entered, in which the user is shown the prototype using familiar data. User suggestions for improvements are incorporated into successive iterations until the working model provides complete satisfaction. It is then suggested that detailed design be derived from the user-approved prototype to adequately document the system for maintenance purposes. Finally, it is recommended that an implementation checklist be used to ascertain that the prototype has been officially approved and that all documentation is complete and accurate.

These are good procedures to follow in rapid prototyping. However, on the basis of the case studies, there are three phases that should be added: planning, as the first phase; modeling, or benchmarking, as a phase to follow prototype iteration; and training of system maintenance staff. Figure 4 shows all phases of the new, revised life cycle model, with appropriate feedback and feed-forward of information indicated by arrows. The reasons for the addition of the three new phases should be apparent from the following material.

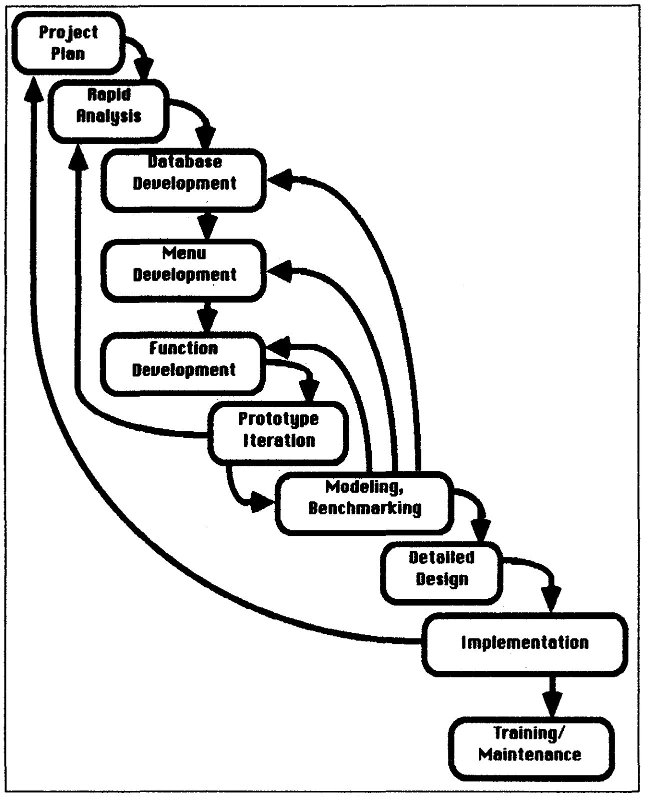


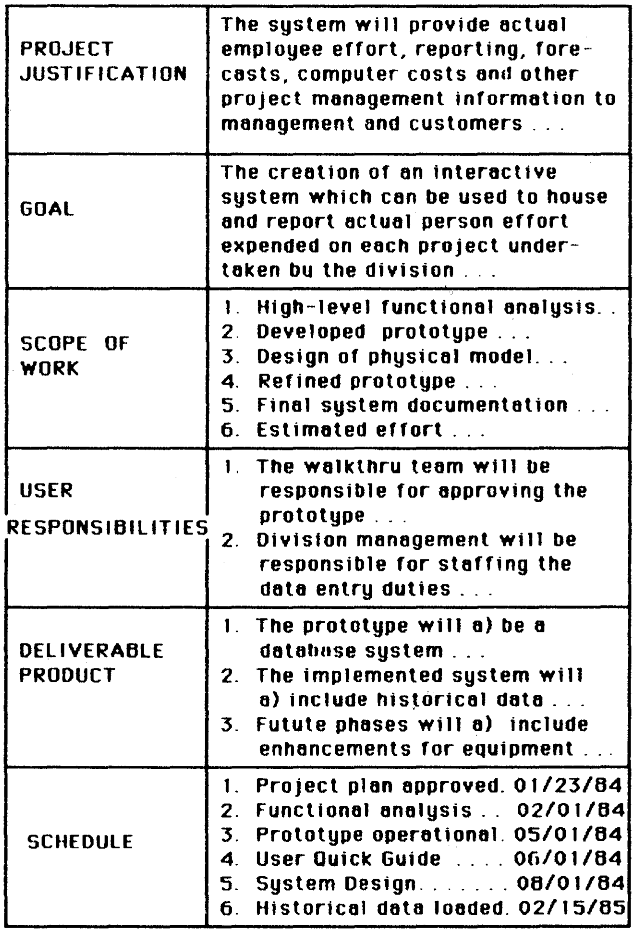
Figure 4—Rapid Prototype Life Cycle

A COMPARISON OF LIFE CYCLE ACTIVITIES

Planning the Failure

A project plan, partially shown in Figure 5, was produced. It was correct in format and approved by a walkthrough team consisting of peers. In reality, the project plan was not followed. The budget was cut drastically, both in staff and in schedule. For example, whereas the project plan forecast a demonstrable prototype in October and historical data loaded by experienced staff members, in actuality the prototype was demonstrated in Janaury and implemented in August, complete with historical data, and only inexperienced student labor was used. Why was the budget so severely modified? There were a number of reasons, and not one of them was due to improper motivation. There was an optimistic belief that just maybe this one time the odds could be beaten because so much was at stake. Management was in an untenable situation: an internal system was needed, but proper staffing was difficult to justify, since the new system would have an unclear payoff, at least in the early stages. Student employment was available only for the summer, combined with only one staff member who was terminating at the same time. The belief, the real hope, was that some of the system could be implemented before the staff melted away. In that way, corporate memory could be preserved, the major functions could be used, and some time could be bought to staff the system properly for completion of implementation and for maintenance.

As has often been documented, cutting budgets is almost always unwise and was probably the single most obvious reason in this case for system failure. How does budget cutting differ in a prototyping environment from the traditional development environment? In several ways. One is that, because visible outputs are quickly available, users and managers are easily seduced into believing that the design/code/test phase as well as the modeling phase can be skimped on. Even mature, experienced management and analysts must be especially careful in realizing that menus and screens are not backed up with detailed design and that reports may be perfectly formatted, using data elements carefully defined, yet still suffer tremendously in the performance area. The point is that, as in the case of the failure, even a correctly prepared project plan that is not adhered to will accomplish nothing.



PROJECT JUSTIFICATION	The system will provide actual employee effort, reporting, forecasts, computer costs and other project management information to management and customers . . .
GOAL	The creation of an interactive system which can be used to house and report actual person effort expended on each project undertaken bu the division . . .
SCOPE OF WORK	1. High-level functional analysis. . 2. Developed prototype . . . 3. Design of physical model. . . 4. Refined prototype . . . 5. Final system documentation . . . 6. Estimated effort . . .
USER RESPONSIBILITIES	1. The walkthru team will be responsible for approving the prototype . . . 2. Division management will be responsible for staffing the data entry duties . . .
DELIVERABLE PRODUCT	1. The prototype will a) be a database system . . . 2. The implemented system will a) include historical data . . . 3. Futute phases will a) include enhancements for equipment . . .
SCHEDULE	1. Project plan approved. 01/23/84 2. Functional analysis . . 02/01/84 3. Prototype operational. 05/01/84 4. User Quick Guide . . . . 06/01/84 5. System Design. . . . . . . 08/01/84 6. Historical data loaded. 02/15/85

Figure 5—Project Plan

Planning the Success

A project plan was completed. It was actually improved upon, because the schedule was originally based on traditional development methods calling for a much longer development cycle than was needed for rapid prototyping. The original estimate was 50 person-years; the projected estimate now that one particular phase has been successfully prototyped and implemented is 10 person-years. Since the original plan called for detailed design to be derived from the approved prototype, the design was completed despite arguments that it was now unnecessary because the prototype worked so well. The prototype was not implemented without carrying out all the steps called for in the project plan. It is obviously important not just to prepare a project plan, but to follow the plan during development.

Rapid Analysis: the Failure

The analysis of the system was not terribly difficult, because the system under development was based on an existing system. The existing system was clumsy to use and difficult to maintain because it was old, little analysis had been performed on it when it was first developed, it was COBOL/sequential file/ISAM file/batch based, and it suffered from hard-coded data. It produced only fragmented information for a small segment of the user community. Over the years, the users had developed definite opinions about what their needs really were, where the functions of the system should be expanded, and how the input/output cycles should work within the framework of corporate policy. The rapid-analysis phase proceeded with ease; and although no product can be considered perfect, it probably would look very much the same if it were redone with different analysis staff, just because the strengths and weaknesses of the old system were already known. A paper model was produced, using dataflow diagrams for the system context and essential functions models. The paper model included an entity relationship diagram showing relationships among aggregations of data elements or entities.

This high-level design was probably adequate. The fact that the system would become a failure was not apparent during the Rapid Analysis phase. The paper models were useful during prototyping and the database structure, and functional development followed the design of the models.

Rapid Analysis: the Success

The analysis phase of the successful prototype was quite problematic. First, the system concept was so new that neither users nor developers could offer a clear definition of system requirements. There were some vague notions of what an ideal system would do, but serious misgivings about how many of these functions it would be practical to implement. Second, none of the persons who were performing the analysis had any formal training in structured analysis. Finally, the walkthroughs of the analysis documents were very poorly structured and frequently degenerated into all-day oral defenses of the document by its author.

The result was that when users finally had a chance to review the analysis document, they became confused and angered. They were confused because the paper model did not clearly represent the proposed system requirements. They were angered because what they could understand implied that the system to be developed would be of little use to them. Fortunately, prototype development was under way by the time the ensuing argument between users and analysts reached its peak. Just as the user group was beginning to clamor for development funds to be cut, a demonstrable prototype was ready. When the users saw the demonstration of the prototype, they were pleasantly surprised to discover that a potentially useful and readily modifiable working model had been developed so quickly.

This is actually an example of the proper application of rapid-prototyping techniques. The prototype development team interviewed the analysis team to discover what was really intended in the paper model and to OK changes that users seemed to want. A first-cut working model was then produced within a few weeks. Figure 4 shows feedback from prototype iteration to the rapid analysis phase. In this case that is what happened: Since the prototype received user approval and the paper model did not, the paper model was revised to agree with the prototype.

Prototype Development: Both Cases

Prototype development consists of three steps: database development, menu development, and function development. These steps were carried out in similar fashion for both systems. First a basic database structure was developed on the basis of the paper models. Next a menu structure was created as a user interface, using a screen-based forms tool provided by the relational database management system. Finally, functions driven by the database system were developed. In both cases the writing of third-generation programs was avoided, and a working model was available very quickly.

It is difficult to make unrecoverable mistakes during these phases. The prototype is based on a relational database and has no lines of code, and it is therefore very easy to modify. In fact, since the main purpose is to find out whether the working model pleases the user, many changes are to be expected. That being the case, it is not possible to prove that one prototype is better than another. They are all simply first cuts—intended to be refined, enhanced, tuned, tested, and possibly even thrown away if the original concept proves unfeasible. It is true that in the case of the failure, an inexperienced staff was busy developing a system that would not function very smoothly in a production mode. But then their charter was to produce a rapid prototype, not a production system.

Certainly elements of the failure can be detected in this phase. The quickening pace of the schedule prevented thorough consideration of the volume of data and how those data would need to be linked for outputs. For example, even though data tables are nicely normalized, too many joins may be required in some of the retrieval and reporting functions to make the table structure viable. Elements of the success can also be detected in the prototype development. The developers of the successful prototype realized early in the game that as soon as users saw a useful working model, there would arise a great clamor for its implementation as a production system. Therefore these developers purposely built nonworking functional modules into the system. These modules looked nice during demonstrations but prevented the system from being used in production.

Iterating the Failure

When and how should a prototype be reviewed by users? "Often" is not an adequate answer, as can be seen in the case of the failure. This prototype was turned over to all potential users for hands-on experimentation—even the skeptics had a chance to exercise it. This step was useful for obtaining user feedback about desired changes, but it had some undesirable side effects. The demonstrations served to raise expectations to unrealistic levels. The appearances were that a production system was in place that was being modified to meet true user requirements. Surely this system would be released soon to replace the current one and solve all problems! Much of the goodwill of this user community was lost when the prototype was not immediately implemented, and even more will be lost if it is scrapped.

Since users were told that fourth-generation techniques and rapid prototyping were being used to develop the system, the tendency was to blame these techniques for the failure. It is only in this post-mortem that project management and the incorrect application of rapid prototyping techniques are being considered as possible causes of the failure. The skeptics now say that rapid prototyping means inadequate design and slipshod testing, and they advocate the return to traditional software development. Therefore, the failure of one rapid prototype in a critical application may mean that all the advantages of rapid prototyping will be lost to future users and developers.

Iterating the Success

The successful prototype was never released to users in a hands-on mode. Demonstrations were provided where developers operated the prototype and users were simply the audience. The only people who were allowed to use the prototype were the developers themselves. Large groups of users reviewed the prototype in several of these demonstrations, and their comments were used to refine the system. Thus, by giving users the feel of how useful the implemented prototype would be, the developers obtained user feedback without raising expectations too high. When the occasional user demanded immediate implementation, the nonworking modules mentioned earlier effectively prevented it from happening.

The result of this rigid stance on the part of the developers was that implementation was delayed until a smoothly operating, bug-free system could be delivered. Since this was accomplished in much less time than was previously thought possible, rapid prototyping acquired a reputation for being a better way to develop application systems. High-level managers within the company are now proud to claim that Company B has mastered this state-of-the-art technique.

Modeling or Benchmarking

This is one of the newly recommended phases. It is possible that prototype iteration has provided functionality with which users are perfectly satisfied, but the system still won't work. The modeling phase consists of performance testing to determine response time characteristics of the system in a production environment. It does not involve user interaction, but it can be critical for many different types of applications. The system must be exercised with expected maximum usage and data loads before it can be determined that the current incarnation is implementable. This is particularly true of relational database systems, where it is possible to create database structures and functional procedures that will work adequately under small loads but slow down unacceptably under large loads.

There is not much to say about this phase in the case of the failure; it simply didn't happen. The prototype was implemented, and its undesirable performance characteristics were discovered during production operations. On the other hand, the successful prototype was used by developers in a realistic production emulation mode to actually perform all its functions on large amounts of real data. As a result, the successful prototype went into a rather lengthy phase of elaborate tuning and refinement. Tables were radically restructured, and some procedures that had been developed rapidly with fourth-generation techniques were replaced with modules developed using lower-level, but more efficient, procedures. In this way a pleasing demonstration-only prototype was converted into a production-quality system.

Designing the Failure

The failure did not go undocumented. A very professional, accurate, and understandable user guide was produced. This was accomplished with relative ease by printing input and output screens and menus and adding explanatory dialogue. Missing from the final documentation was a detailed design of system procedures and control flows. High-level analysis is fine for prototyping, but next to useless for maintenance work or system tuning.

For the system to be easy for anyone to maintain, detailed descriptions of its architecture and control sequences need to be published. Unfortunately, this was one of the steps that was skipped in the rush to implement. Later, when it was discovered that there were some problems in running the prototype in a production mode, these problems were not easy to remedy without the proper design documentation. If the project plan had been adhered to, it would have been noted that an extensive detailed design phase was called for. Remember that the project plan was agreed upon by users, managers, developers, and their peers. The organizational wisdom at the time had determined that the design effort called for was absolutely critical to project success.

Designing the Success

The data that were entered into the successful prototype and used to produce output consisted of detailed specification of the prototype itself. All functional modules were completely specified to the lowest levels, complete with accurate data definitions. This detailed design was derived from the latest user-approved version of the prototype. During the last part of the benchmarking phase and into implementation, a change in development staff personnel made this documentation extremely valuable. Since the specification now accurately describes the implemented system in complete detail, a new maintenance person can learn the system well enough to make needed changes in less than a week.

Implementing the Failure

It is surprising that the failure was allowed to be put into production without the developers' first completing an implementation checklist, considering that the organization where the failure occurred usually adheres strictly to rigid implementation standards that explicitly call for such a checklist. Perhaps, since this system was developed at the request of internal users rather than external customers, it was thought that this was a step that could be eliminated in the interests of desired budget cutting. Implementation checklists are simply a way of determining whether the project completed all activities and delivered all products called for by the project plan and internal standards. Since the failure clearly did not, publishing an implementation checklist would have advertised this embarrassment.

Implementing the failure consisted of an abrupt switch from use of the old system to use of the prototype. This is the point at which those responsible for maintaining the prototype began to complain loudly and bitterly. They said that the table structures were all wrong, that coding standards had been violated, and that the lack of adequate design made modifications next to impossible. Poor performance characteristics of the prototype gave them all the ammunition they needed to get the prototype scrapped and return to using the old system.

Implementing the Success

The successful prototype was also implemented without benefit of a formal checklist. The organization simply does not use such a device, relying instead on elaborate review procedures during earlier development phases. At the point where the system test plan has been reviewed and approved and the proposed system subjected to it successfully, the system is delivered to the customer. In this case, however, the prototype developers kept their own checklist and refused to let the system be implemented until it had satisfied all the criteria defined in the project plan for successful implementation.

Maintaining the Failure

The maintenance phase was the most difficult in the case of the failure. It will be one of the causes in the case of the prototype's final demise. The maintenance staff personnel were not familiar with rapid prototyping techniques and had been loudly skeptical, all during development, of the advisability of using these techniques. Now they had an opportunity to say "I told you so." Courses offering instruction in rapid prototyping techniques had been offered within the organization, but later discontinued. Previous graduates of these courses were available, but none of them was assigned to maintain this prototype. Management's thinking may have been that, although training may be helpful for developing rapid prototypes, it is not needed for maintaining them.

A rapid prototype involves the application of fairly radical techniques incorporating fourth-generation language and relational database tools. Maintenance personnel with a third-generation coding background will be hard pressed to understand a system developed with these techniques unless they are given adequate training. In a highly technical field, lack of understanding is threatening to one's professional stature; there is a tendency to belittle techniques with which one is not familiar.

Maintaining the Success

Even though the staff responsible for implementing and maintaining the system consisted of personnel completely different from the development staff, they had been thoroughly trained in and believed in the techniques of rapid prototyping. Remember that only one interim product from an extensive proposed tool suite was implemented. The remainder of the tools will be developed in rapid prototype fashion by the current staff. They are enthusiastic because they have discovered that the modular style of rapid prototype development leads to reusable and replaceable functional modules. This will make their future development work much easier than the development of the original prototype.

Examining the Critical Differences

Figure 6 lists the differences between the life cycle activities of the two projects. For each area where a difference occurred, there is a brief descriptive statement. It can be observed that there are no significant differences applicable to the development of the prototypes themselves. The conclusion that can be drawn is that one project was managed well and the other was mismanaged. In addition, there are also some obvious techniques for making a prototype successful, rather than for making a good prototype.

CONCLUSIONS DRAWN

One might ask, in the case of the failure, why the mature project team members deviated so far from years of experience-based knowledge. The answer is an important one; prototyping is still new as an industry tool, and we are prone to misuse it. It is the most valuable of all of the tools of communication between customer and developer. It solves the problem of delivering the wrong system better than anything else we know of at this time. It is, however, potentially hazardous because it can persuade all of us, both customer and developer, that now that we have successfully defined what is desired and necessary in a system, implementation will be trivial. While identifying the problems is a large part of the battle in any problem-solving situation, it is by no means the entire activity.

Life Cycle Phase	Failed Prototype	Successful Prototype
Project Planning	Adequate, but not followed	Followed precisely
Rapid Analysis	Well done	Poorly done, but revised to agree with prototype
Database Development	Adequate	Adequate
Menu Development	Adequate	Adequate
Function Development	Full functionality of all specified processes	Included non-functioning diplay processes
Prototype Iteration	Hands on for all users	Hands on for developers only; demo to users
Modeling/ Benchmarking	Not done	Extensive exercise and tuning of prototype
Detailed Design	User Guide + high level paper model	User Guide + detailed physical model specs
Implementation	No checklist, not done according to plan	No checklist, but done according to plan
Training/ Maintenance	No training, sceptical staff	Adequate training, enthusiastic staff

Figure 6—Life Cycle activity: Case Comparison

Defining Rapid Prototyping

During the prototype iteration phase, functions may be redefined; menus may be added, deleted or modified; and, more likely than not, the database structure may be altered (although this activity will probably peak during modeling when restructuring is done for performance reasons). Once again, the point must be made that the system is not fully fleshed out at this point and should not be implemented. In the case of the failure, a great deal was lost by putting the system on the floor for everyone to use and test. The skeptics criticized every minor problem as though it were a fully tested system instead of a refined prototype, and the proponents liked it so much they were incredulous that the system might be scrapped. Much goodwill was sacrificed in both camps because *prototype* was improperly defined for the entire customer base and because there was no "friendly user phase."

Following a Sensible Plan

A sensible project plan is one that includes all the activities listed in Figure 4 with realistic estimates of the time required to complete them. A peer review of the plan will help to assure reasonableness. If users or management want to revise the plan or stray from its directives, a new plan with new deliverables should be published. If the project plan was deemed reasonable at the start of a project, budget cuts during development must result in reduced deliverables if implementation is to be successful. If the prototype development team is unsure that the current project plan is being adhered to, they should insist on the publication of a checklist prior to implementation.

Training for Rapid Prototyping

In the case of the failure there was a broad-based misconception on the part of many users, managers, and proposed maintenance staff about the definition and meaning of *rapid prototyping*. To reiterate the common confusion, rapid prototyping is a requirements analysis technique, not a system development technique. These misconceptions also exist in the organization that implemented the successful system; but they did not become critical because they were not held by key managers, developers, or maintenance staff.

The best way to clear up misconceptions about rapid prototyping is with an in-house training program. Unfortunately, such training is not readily available from commercial sources. A possible method of implementing such a program within an organization is to select a successful rapid prototyper and place him or her in charge of designing the training program. If no such person is available, then it might be wise to reconsider the advisability of using rapid prototyping techniques. The misuse of rapid prototyping, as illustrated by the failure, can yield more disadvantages than the benefits to be gained from its proper application.

REFERENCES

1. Brice, L., J. Connell, and D. Shafer. "Using INGRES as a Rapid Prototyping Device During Development of Management Information Applications." *IEEE Proceedings of the Symposium on Application and Assessment of Automated Tools for Software Development.* Silver Spring, Md.: IEEE Computer Society Press, 1983, pp. 34–43.
2. Connell, J., and L. Brice. "Rapid Prototyping." *Datamation,* 30, (1984), pp. 93–100.

AGENT: An advanced test-case generation system for functional testing

by ZENGO FURUKAWA and KENROKU NOGI
Systems Development Laboratory, Hitachi Ltd.
and
KENJI TOKUNAGA
Software Works, Hitachi Ltd.
Kawasaki, Japan

ABSTRACT

This paper presents the AGENT method, a systematic test-case generation method for functional testing. AGENT consists of two steps. In the first step, a functional specification of a program is described with a formalized notation called a *function diagram (FD)*. An FD consists of two components, a state transition which is described with a state transition diagram, and a set of boolean functions which are described with a cause-effect graph or a decision table. In the second step, test cases are mechanically generated from the FD.

Test cases generated by this method satisfy the following conditions: (1) They validate input conditions and output conditions in all states, and (2) They pass all transitions at least once and include a case of bypassing and getting through each loop in a structured state transition.

INTRODUCTION

There are two important subjects for software testing techniques, the improvement of testing efficiency, and the improvement of testing precision. The latter is more important because the purpose of software testing is the detection of errors. There are two types of testing, functional and structural.[1] Functional testing is thought to be more basic than structural testing because errors in a program are defined as *behavior discordant* with the functional specification of the program.[2] Nevertheless, functional testing has not been studied sufficiently. One conventional approach to functional testing is to have people read a functional specification and pick out test cases intuitively. However, high software reliability cannot be achieved by this method.

We have developed AGENT (Automated GENeration method of Test-cases), a systematic test-case generation method for functional testing. AGENT is composed of the following components:

1. A function diagram (FD) which formally expresses the functional specification of a program.
2. A mechanical procedure for generating test cases from the FD.

An FD model is composed of a state transition model and a boolean function model. A state transition expresses input data sequences and corresponding output data. A boolean function in a state expresses the correspondence between conditions on input and output data. A test case constitutes a sequence of states passed through in testing and a pair of conditions in each state which must be satisfied by the input and output data. AGENT automatically generates test cases from an FD.

AGENT has the following characteristics:

1. Description of a function specification with an FD is easier than with a boolean function alone because an order of input data need not to be changed into the constraints needed for a boolean function.
2. Criteria for test case generation are clear, and the number of test cases generated is reasonable.
3. Test cases are automatically generated from an FD.

We had previously developed the AGENT-I[3] program, which generates test-cases from a cause-effect graph (CEG).[3] However, since it was difficult to describe a functional specification with CEG, we later developed the FD notation and the AGENT-II program (usually called merely AGENT).

This paper primarily discusses the method used for generating test cases. It explains FD notation and the generation algorithm, and provides a concise outline of the AGENT program.

FUNCTION DIAGRAM (FD)

Some studies have been done on the description of a functional specification as a dependency between input and output data. A cause-effect graph[1,2] expresses the correspondence between input and output data based on boolean function, and the interdependency among input data owing to constraint conditions.

A functional specification of a program usually consists of a dynamic and static specification (see Figure 1). The former expresses the order of input data (or translation order); the latter, a correspondence between input and output conditions. A specification described only by a static specification is impractical because of the large number of possible combinations. A dynamic specification must be used to simplify the functional specification. A state transition model is suitable for describing a dynamic specification. In a state transition model, the output data and subsequent state are determined by the input data and the present state. A boolean logic model is suitable for describing a static specification. In a boolean logic model, the output data are determined only by the input data.

Function Diagram Notation

The notations shown in Figure 2 are used to describe a function diagram as follows:

1. *State transition.* A state transition is described with states and transitions.
   (a) *State.* A state indicates a place (or time) wherein data are input. An initial state is the starting point of activity and final states are possible ending places of activity.
   (b) *Transition.* A transition designates a change of a state. A state prior to transition is called a *tail state,* and after transition, a *head state.* The boxed T symbol over an arrow expresses a boolean function with decision table or cause-effect graph. If boolean function is simple, then the input and output condition is written directly on the arrow.
2. *Boolean function.* A boolean function in each state is described with a cause-effect graph or decision table. Interdependencies among input conditions are expressed with constraint conditions.

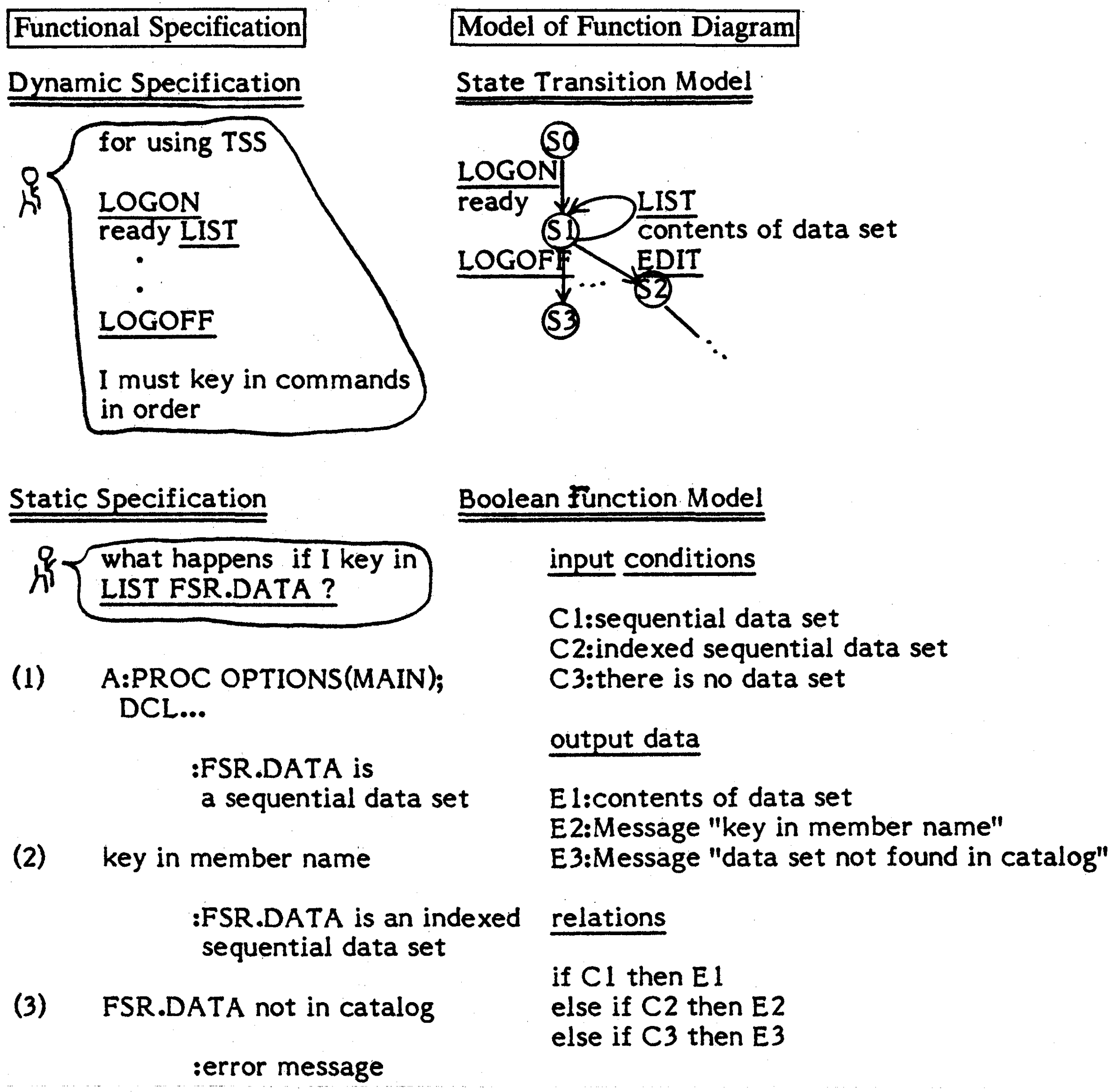
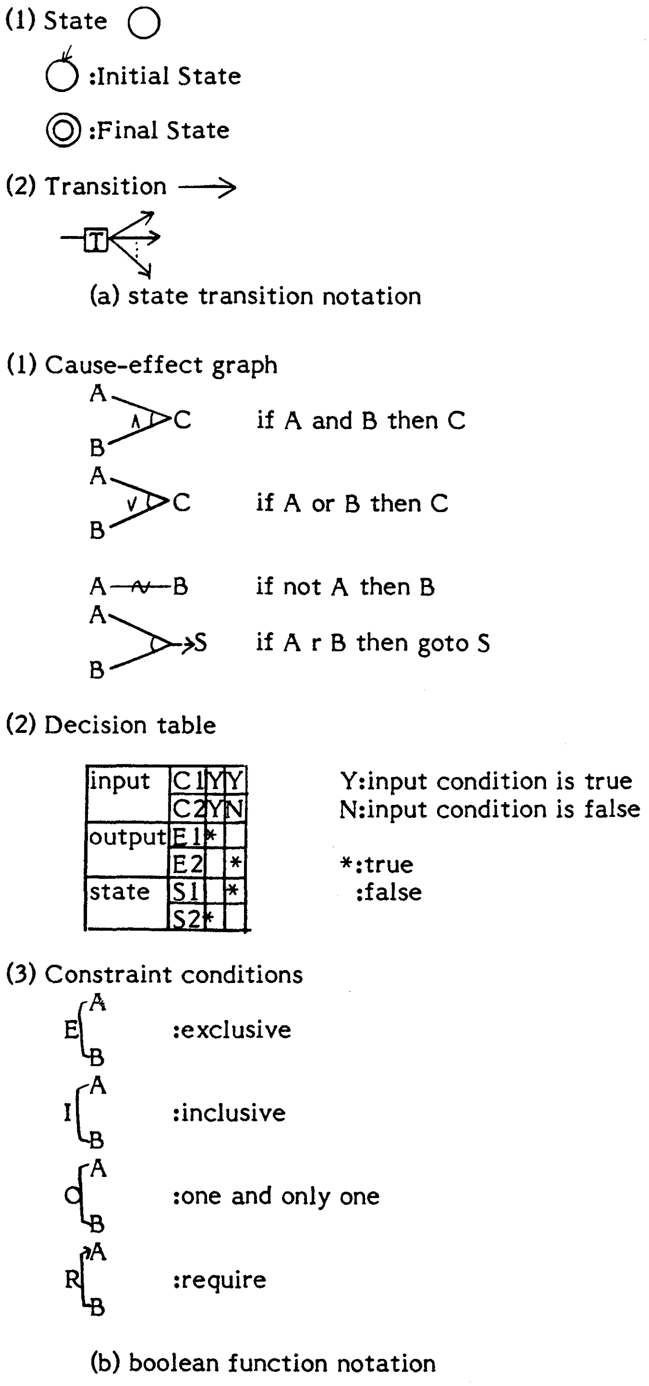


Figure 1—Model of a functional specification

An Example of a Function Diagram

Figure 3 is a sample function diagram representing the following functional specification of a simplified Automatic Teller Machine (ATM). State S3 represents the display *insert card.*

1. The ATM displays *key in pass code* when a cash card is inserted.
2. The ATM checks the correspondence between the keyed-in pass code and the code on file. If they are same, the ATM displays *key in sum;* if not, the ATM checks whether the number of times that the correct code has not been entered is equal to three. If so, the ATM displays *stop process,* cancels the card registration, and displays *insert card* for the next customer. If not, the ATM displays *key in pass code again.*
3. When an amount is keyed in, the ATM checks whether it is less than or equal to the balance. If the amount is greater than the balance, the ATM displays *key in sum* and waits for the amount to be keyed in again. If the amount is less than or equal to the balance, the ATM pays the money requested, reports the balance and displays *insert card.*



(a) state transition notation

(b) boolean function notation

Figure 2—Function diagram notation

METHOD OF TEST CASE GENERATION

In generating test cases from an FD, it is important that the number of test cases be practical and that the criteria for test case generation be clear. An FD is composed of a state transition and boolean functions. Myers described a test case generation method for boolean functions in which test cases are generated from a cause-effect graph.[2] The AGENT-I program[3] was developed based on this method. The criteria considered for a state transition is the path testing strategy of programs[4] where a state is substituted for a node and a transition is substituted for a branch.

Using clearly defined criteria, a practical number of test cases are generated from an FD by combining test cases of a state transition (testing paths) with test cases of boolean functions (partial test cases).

Test Case Generation Criteria

Partial test cases which are generated from the boolean functions of an FD include true and false cases of input data conditions, and are nearly minimum.[4] The number of test cases increases linearly with the number of input data conditions.

There are many criteria for state transitions[5]; among these, *passes all states* (C0 coverage) and *gets through all transitions* (C1 coverage) are well known. For the state transition shown in Figure 4(a), testing paths (i.e., sequences of states) of (b) and (c) are example sequences for all states *passing* and all transitions *getting through,* respectively.

In programming, a case of bypassing a loop is apt to be overlooked or a mistake made concerning a loop termination condition. Testing paths should include both a case of bypassing a loop and one of getting through a loop in a state transition. A set of testing paths may satisfy the criteria for all transitions getting through, but not include a case of bypassing a loop. For example, in Figure 4, testing paths (c) get through all transitions, but a case of bypassing the loops at state S2 is not included. It is then necessary to include such a case for testing criteria as in (d).

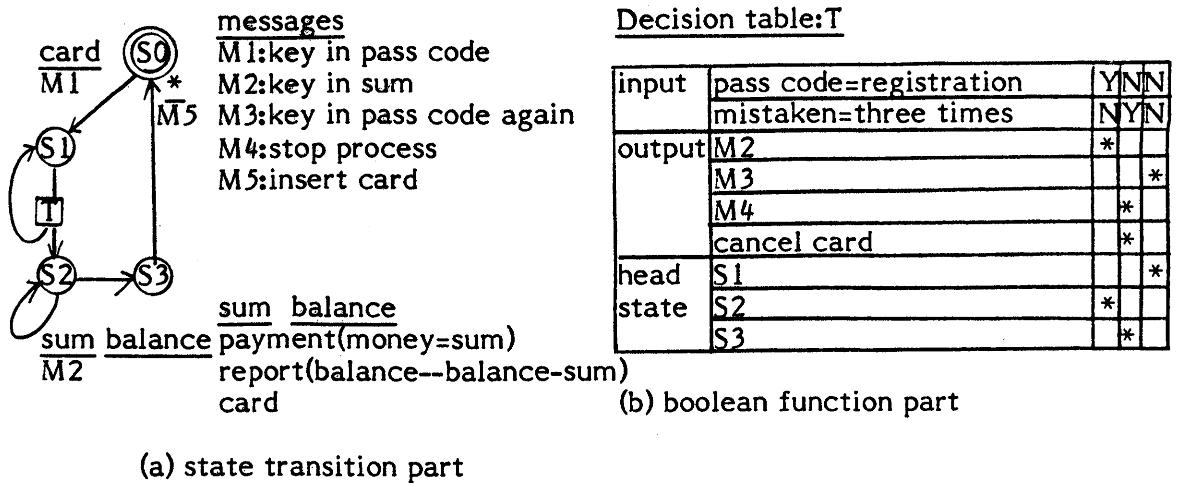
It is difficult to recognize all of the loops of a typical state transition, but easy to recognize them in a structured state transition which is composed of only three kinds of forms (e.g., sequence, selection and iteration, as shown in Figure 5.) In Figure 6, (b) is an example of a structured state transition and of the state transition in (a). There are two loops in Figure 4(a): (S4, S5, S4) and (S4, S5, S3, S2, S4).

For testing path generation, the test cases must pass all transitions of a structured state transition (SST) at least once, and include the cases of bypassing and getting through loop. Figure 4(c) is an example of testing paths which satisfy this criteria. Were the foregoing loop requirements not included in our criteria, they would be satisfied by path 1 and path 2. These do not include the case of bypassing the loops at state S4.

Test Case Generation Procedure

Test cases are generated from an FD by the following procedure.

1. *Generation of partial test cases.* In each state, partial test cases are generated from a cause-effect graph in which causes are input data conditions and effects are output data or head states of the boolean function. If a decision



messages
M1:key in pass code
M2:key in sum
M3:key in pass code again
M4:stop process
M5:insert card

payment(money=sum)
report(balance←balance-sum)
card

(a) state transition part

Decision table:T

input	pass code=registration	Y	N	N
	mistaken=three times	N	Y	N
output	M2	*		
	M3			*
	M4		*	
	cancel card		*	
head state	S1			*
	S2	*		
	S3		*	

(b) boolean function part

Figure 3—An example of FD (ATM)

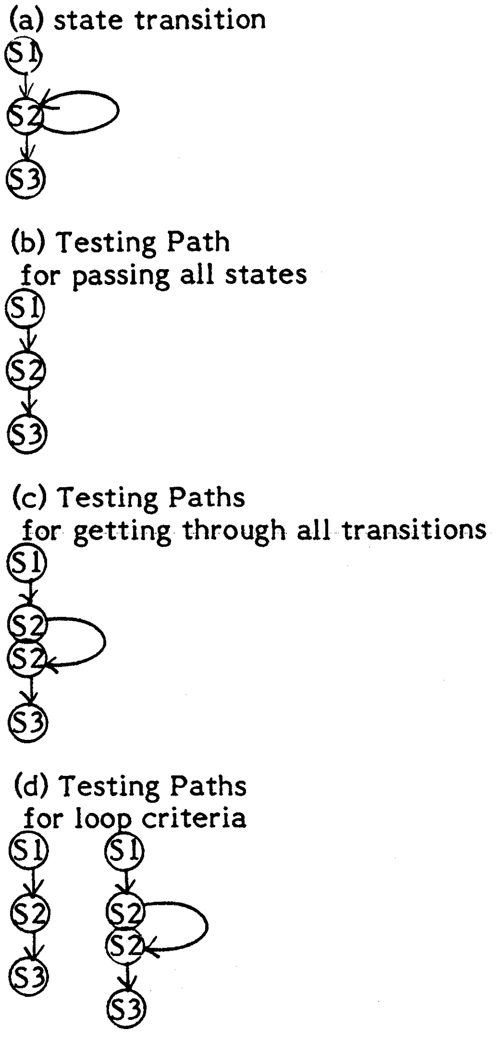


Figure 4—An example of testing paths

table is used to describe the boolean function, it is translated to the equivalent cause-effect graph. A partial test case is composed of a combination of cause values (input conditions) and a combination of effect values (output data or states) which corresponds to a combination of cause values. Only one of the effects which correspond to the states must be true in each partial test case.

2. *Testing path generation.* A state transition is translated to a structured state transition (SST) composed of the three kinds of forms shown in Figure 5(a) by means of a translation from a transition matrix to an expression[6] composed of the three kinds of operations shown in Figure 5(b). A testing path is generated from the initial to the final state with the criteria in the previous section always being satisfied.
3. *Synthesis of test cases.* Test cases of an FD are synthe-

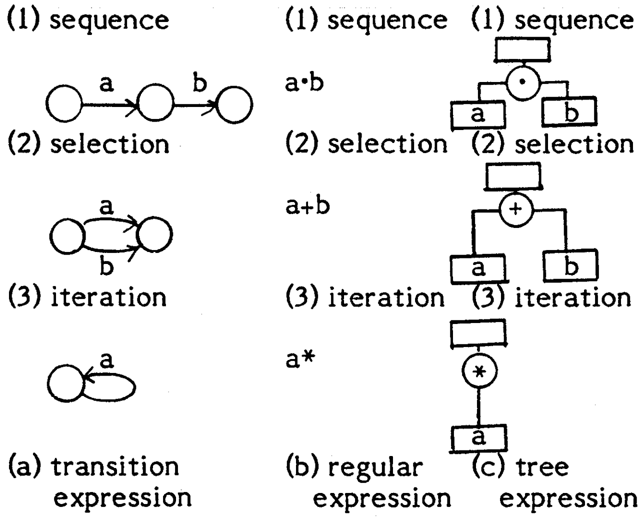


Figure 5—Notation for a structured state transition

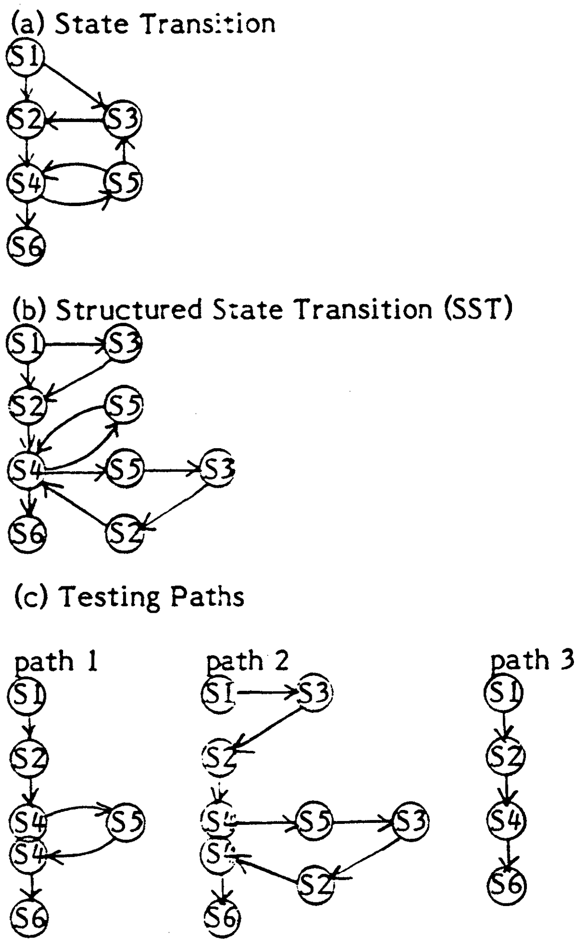


Figure 6—An example of SST and testing paths

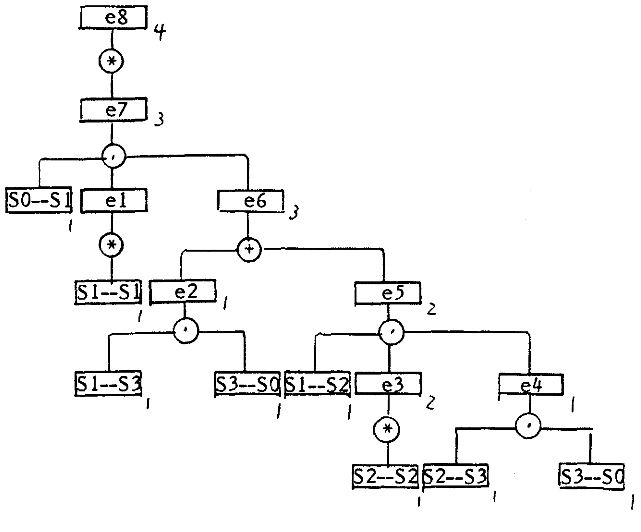


Figure 7—An example of a condition structure tree

sized from the testing paths and partial test cases in each state of the FD. They are made up of sequences of states from initial to final and combinations of input data conditions and combinations of output data in each state. In the synthesis algorithm, a partial test case is assigned to a state of a testing path in which a head state of the partial test case is the same as the next state of the assigned state in the testing path.

Synthesis Algorithm of Test Cases

An SST is expressed as described by the tree expression shown in Figure 5(c) for test case synthesis. A tree expression of an SST is called a condition structure tree (CST). The CST of the Automatic Teller Machine (ATM) is presented in Figure 7.

Test cases are synthesized according to the following steps:

1. The number of partial test cases which have the same head state is counted in each state, and the sum is set for the corresponding leaf of the CST.
2. Each node of the CST is retrieved in post order, and the number of test cases in each node is calculated with the rules shown in Figure 8.
3. Test cases are synthesized in numerical order from 1 to n where n is the number of test cases in the root node of the CST. The nth test case in each node is made up from children's test cases by the following rules (where c(i) is the number of test cases of the ith child node.
   (a) *Sequence.* In the ith child node, the jth test case is

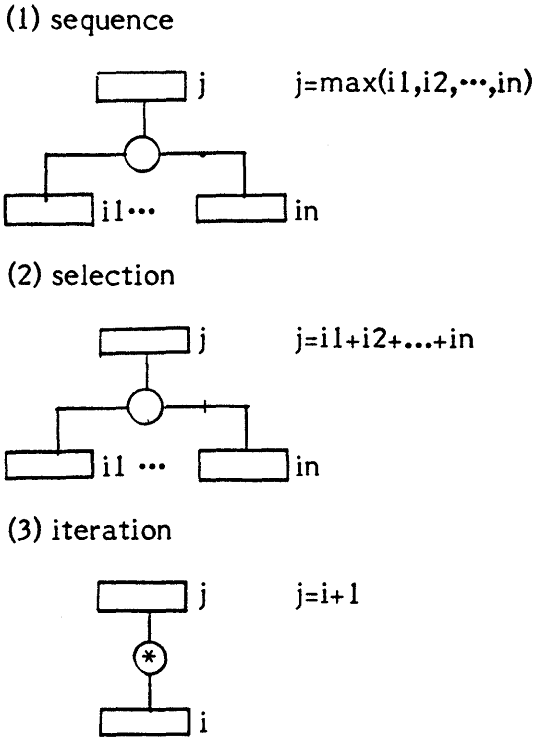
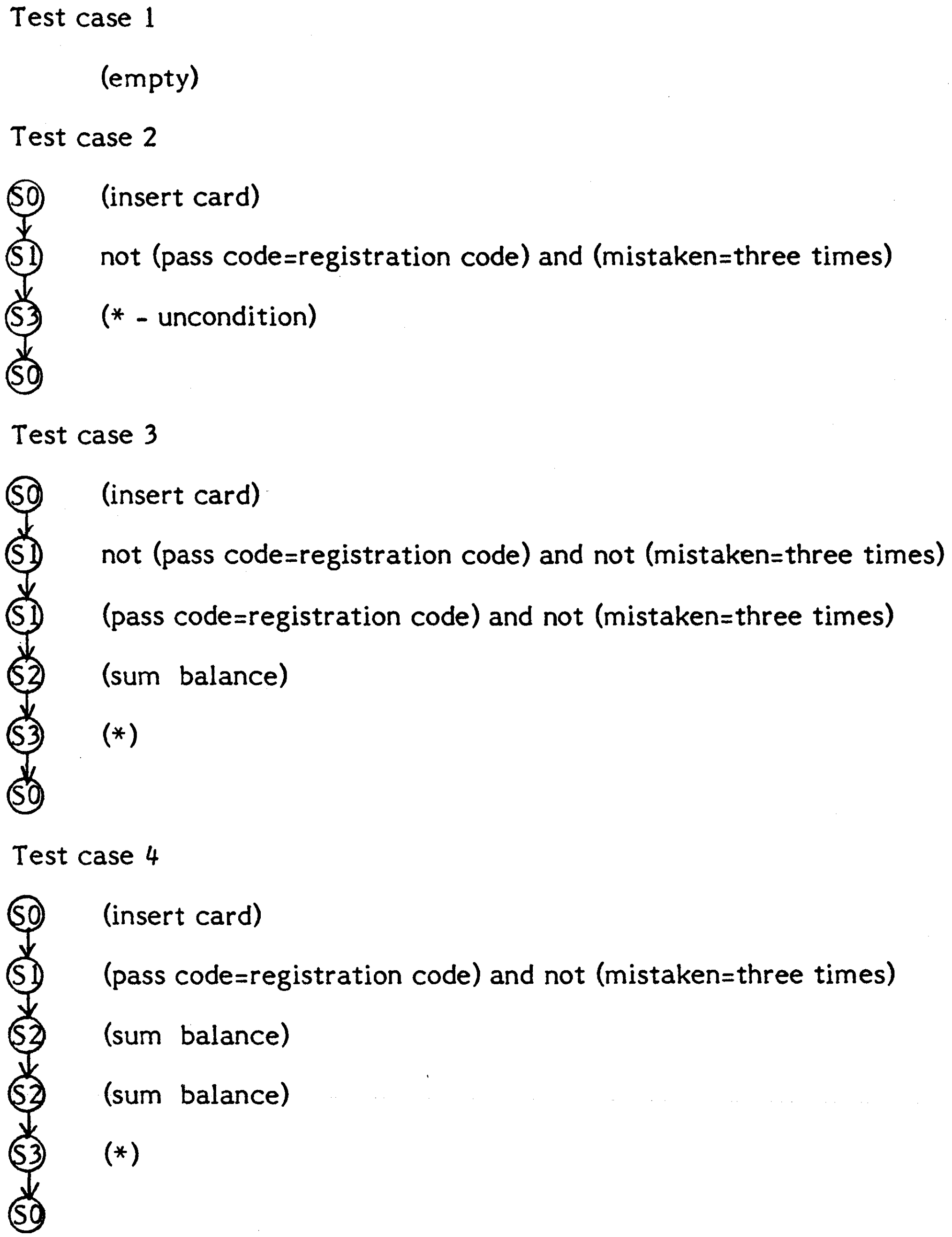


Figure 8—The rule for counting the number of test cases



Test case 1

(empty)

Test case 2

(insert card)

not (pass code=registration code) and (mistaken=three times)

(* - uncondition)

Test case 3

(insert card)

not (pass code=registration code) and not (mistaken=three times)

(pass code=registration code) and not (mistaken=three times)

(sum balance)

(*)

Test case 4

(insert card)

(pass code=registration code) and not (mistaken=three times)

(sum balance)

(sum balance)

(*)

Figure 9—An example of test cases

extracted and test cases for all children are connected. j satisfies (1-1)

$$j(i) = \text{mod}(n - 1, \text{c}(i)) + 1 \qquad (1\text{-}1)$$

where $1 \leq i$ the number of children.

(b) *Selection.* The kth test case of the jth child node is extracted. j and k satisfy (1-2) and (1-3) respectively.

$$\sum_{m=1}^{j-1} \text{c}(m) < n \leq \sum_{m=1}^{j} \text{c}(m) \qquad (1\text{-}2)$$

$$k = n - \sum_{m=1}^{j-1} \text{c}(m) \qquad (1\text{-}3)$$

(c) *Iteration.* The jth test case of the child node is extracted. If j equals zero, then the test case is empty.

$$j = \text{mod}(n - 1, \text{c}(1)) \qquad (1\text{-}4)$$

(d) *Leaf.* A tail state of a transition which corresponds to a leaf and the jth partial test case is extracted.

$$j = \text{mod}(n - 1, \text{the number of partial test cases}) + 1 \qquad (1\text{-}5)$$

A test case is composed of a sequence of tail states which are extracted at the leaves, and combinations of input conditions and output data which are parts of the partial test cases at the leaves.

The numerals on the right side of the CST node box in Figure 8 represent the number of test cases of each node. The ATM example has four test cases. Each test case is described in Figure 9.

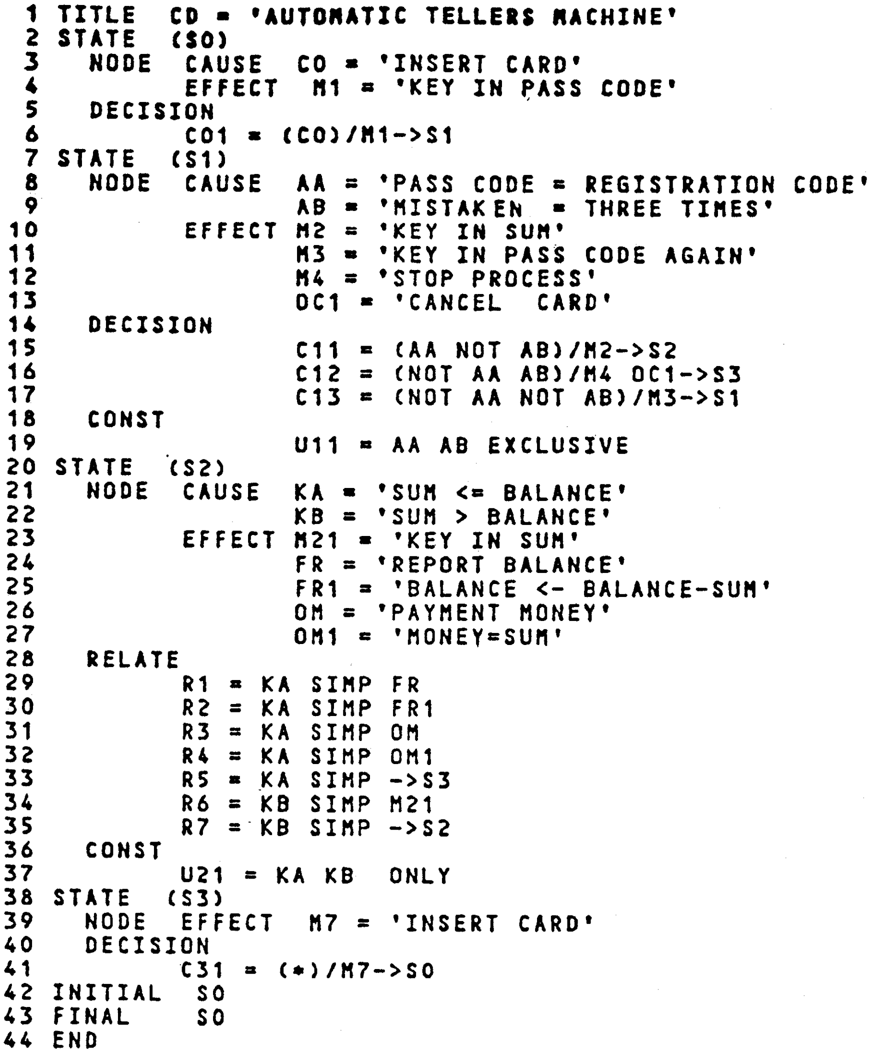
THE AGENT PROGRAM

The AGENT program, which automatically generates test cases from an FD, was written in PL/I and put into operation in 1983. The functions of the AGENT program and some experiences with it are briefly described below.

Input and Output

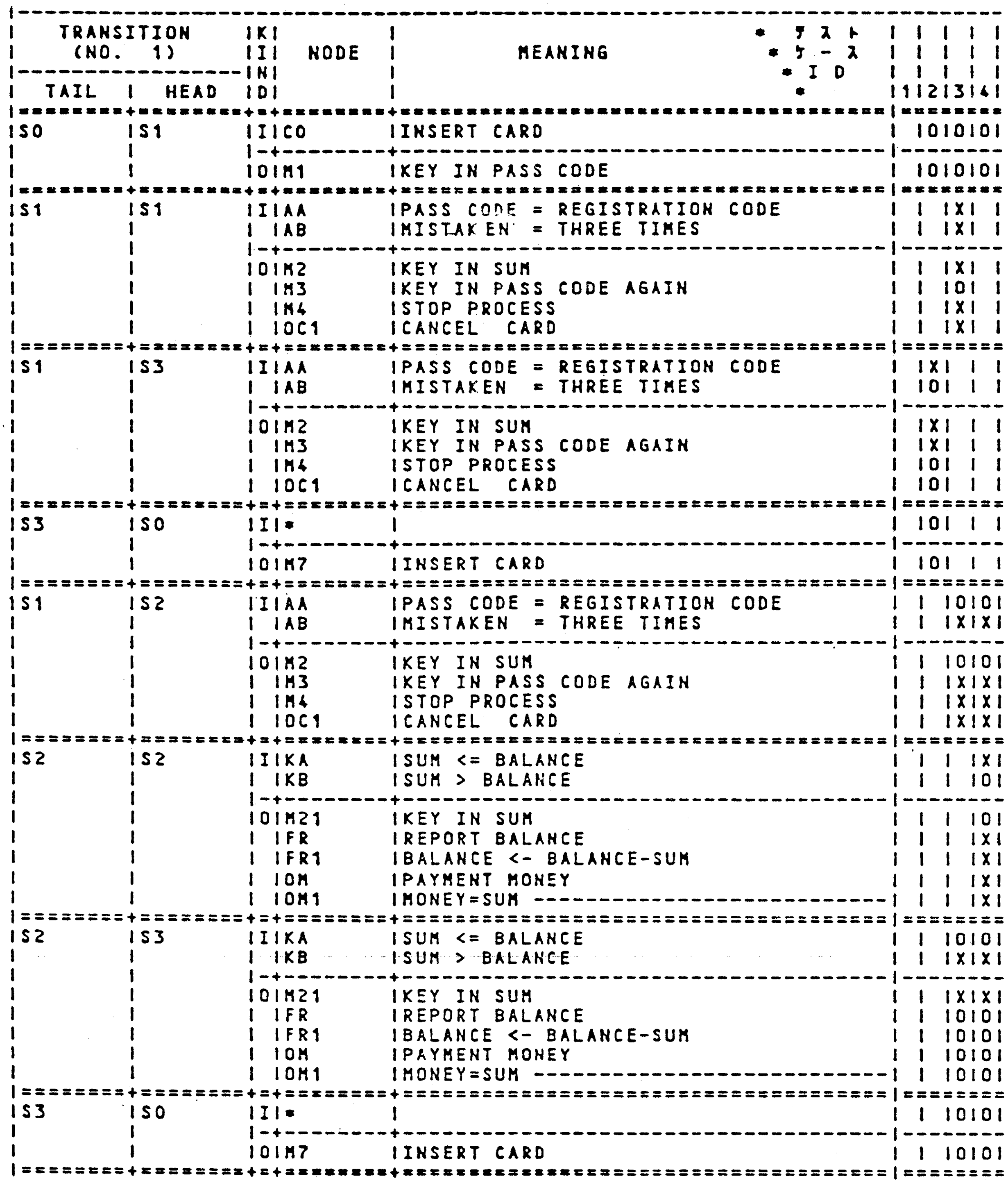
The source list, test case table and test case sheet of the ATM example are presented in Figures 10 and 11. The input to the AGENT program is an FD which is written in function diagram language. A function diagram is composed of a title statement (TITLE), state statements (STATE), an initial state statement (INITIAL), a final state statement (FINAL) and an end statement (END). In each state statement, a boolean function is described with condition definitions (NODE), boolean expression (RELATE) or decision table definitions (DECISION), and constraint condition definitions (CONST).

The main output of AGENT is a table of test cases. A source list, decision tables, and a transition matrix can also be output. In a test case table, a transition is composed of a tail state (TAIL), head state (HEAD). Conditions (NODE) in



```
TITLE  CD = 'AUTOMATIC TELLERS MACHINE'
STATE  (S0)
  NODE  CAUSE  C0 = 'INSERT CARD'
        EFFECT  M1 = 'KEY IN PASS CODE'
  DECISION
        C01 = (C0)/M1->S1
STATE  (S1)
  NODE  CAUSE  AA = 'PASS CODE = REGISTRATION CODE'
               AB = 'MISTAKEN = THREE TIMES'
        EFFECT M2 = 'KEY IN SUM'
               M3 = 'KEY IN PASS CODE AGAIN'
               M4 = 'STOP PROCESS'
               OC1 = 'CANCEL  CARD'
  DECISION
               C11 = (AA NOT AB)/M2->S2
               C12 = (NOT AA AB)/M4 OC1->S3
               C13 = (NOT AA NOT AB)/M3->S1
  CONST
               U11 = AA AB EXCLUSIVE
STATE  (S2)
  NODE  CAUSE  KA = 'SUM <= BALANCE'
               KB = 'SUM > BALANCE'
        EFFECT M21 = 'KEY IN SUM'
               FR = 'REPORT BALANCE'
               FR1 = 'BALANCE <- BALANCE-SUM'
               OM = 'PAYMENT MONEY'
               OM1 = 'MONEY=SUM'
  RELATE
        R1 = KA SIMP FR
        R2 = KA SIMP FR1
        R3 = KA SIMP OM
        R4 = KA SIMP OM1
        R5 = KA SIMP ->S3
        R6 = KB SIMP M21
        R7 = KB SIMP ->S2
  CONST
        U21 = KA KB  ONLY
STATE  (S3)
  NODE  EFFECT  M7 = 'INSERT CARD'
  DECISION
        C31 = (*)/M7->S0
INITIAL  S0
FINAL    S0
END
```

Figure 10—An example of a source list



```
|------------------------------------------------------------------------------
|   TRANSITION    |K|         |                              * ﾃ ｽ ﾄ  | | | | |
|   (NO.  1)      |I|  NODE   |             MEANING           * ｹ ｰ ｽ  | | | | |
|-----------------|N|         |                                * I D  | | | | |
| TAIL   |  HEAD  |D|         |                                 *     |1|2|3|4|
|========+========+=+=========+=======================================|========
|S0      |S1      |I|CO       |INSERT CARD                            | |O|O|O|
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M1       |KEY IN PASS CODE                       | |O|O|O|
|========+========+=+=========+=======================================|========
|S1      |S1      |I|AA       |PASS CODE = REGISTRATION CODE          | | |X| |
|        |        | |AB       |MISTAK EN = THREE TIMES                | | |X| |
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M2       |KEY IN SUM                             | | |X| |
|        |        | |M3       |KEY IN PASS CODE AGAIN                 | | |O| |
|        |        | |M4       |STOP PROCESS                           | | |X| |
|        |        | |OC1      |CANCEL  CARD                           | | |X| |
|========+========+=+=========+=======================================|========
|S1      |S3      |I|AA       |PASS CODE = REGISTRATION CODE          | |X| | |
|        |        | |AB       |MISTAKEN  = THREE TIMES                | |O| | |
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M2       |KEY IN SUM                             | |X| | |
|        |        | |M3       |KEY IN PASS CODE AGAIN                 | |X| | |
|        |        | |M4       |STOP PROCESS                           | |O| | |
|        |        | |OC1      |CANCEL  CARD                           | |O| | |
|========+========+=+=========+=======================================|========
|S3      |S0      |I|*        |                                       | |O| | |
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M7       |INSERT CARD                            | |O| | |
|========+========+=+=========+=======================================|========
|S1      |S2      |I|AA       |PASS CODE = REGISTRATION CODE          | | |O|O|
|        |        | |AB       |MISTAKEN  = THREE TIMES                | | |X|X|
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M2       |KEY IN SUM                             | | |O|O|
|        |        | |M3       |KEY IN PASS CODE AGAIN                 | | |X|X|
|        |        | |M4       |STOP PROCESS                           | | |X|X|
|        |        | |OC1      |CANCEL  CARD                           | | |X|X|
|========+========+=+=========+=======================================|========
|S2      |S2      |I|KA       |SUM <= BALANCE                         | | | |X|
|        |        | |KB       |SUM > BALANCE                          | | | |O|
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M21      |KEY IN SUM                             | | | |O|
|        |        | |FR       |REPORT BALANCE                         | | | |X|
|        |        | |FR1      |BALANCE <- BALANCE-SUM                 | | | |X|
|        |        | |OM       |PAYMENT MONEY                          | | | |X|
|        |        | |OM1      |MONEY=SUM -----------------------------| | | |X|
|========+========+=+=========+=======================================|========
|S2      |S3      |I|KA       |SUM <= BALANCE                         | | |O|O|
|        |        | |KB       |SUM > BALANCE                          | | |X|X|
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M21      |KEY IN SUM                             | | |X|X|
|        |        | |FR       |REPORT BALANCE                         | | |O|O|
|        |        | |FR1      |BALANCE <- BALANCE-SUM                 | | |O|O|
|        |        | |OM       |PAYMENT MONEY                          | | |O|O|
|        |        | |OM1      |MONEY=SUM -----------------------------| | |O|O|
|========+========+=+=========+=======================================|========
|S3      |S0      |I|*        |                                       | | |O|O|
|        |        |-+---------+---------------------------------------|--------
|        |        |O|M7       |INSERT CARD                            | | |O|O|
|========+========+=+=========+=======================================|========
```

Figure 11—An example of a test case table

the tail state are divided into those for input data (I) and output data (O). Each test case is expressed in one column of a test case table. A blank indicates that a test case did not get through that transition, an 'O' indicates that a condition is true, and an 'X' indicates that a condition is false.

The first test case in Figure 11 did not get through any transition. Since the ATM example has the same initial and final state (S0), the first test case is a case of bypassing the loops at S0. The second test case gets through a sequence of states (S0, S1, S3, S0) where at each state, input conditions are set and output data are checked. For example, at state S1, the former are to key in error pass code and iterate three times to

TABLE I—Summary of experience

no.	n	L	e	es	T
1	4	1.5	7	8	4
2	2	4.5	1	1	15
3	1	13.0	1	1	12
4	9	3.4	21	59	25
5	2	4.5	3	3	13
6	6	2.5	11	14	13
7	5	3.2	11	11	12
8	9	4.2	21	51	33
9	11	2.7	15	33	22
10	10	18.5	28	37	131
11	7	18.9	28	104	95

n=the number of states in a state transition diagram
L=the mean number of input conditions
e=the number of transitions
es=the number of SST transitions
T=the number of test cases

error and the latter are to certify that the message *key in pass code again* is not output, the message *stop process* is output, and the card registration is cancelled.

Experiences with AGENT

Table I is a summary of AGENT applications. Numbers 1 to 5 are experimental specifications for FD descriptions; numbers 6 to 11 are parts of practical software. The following points have been deduced from the data and user suggestions.

1. *Size of FD.* As shown in Table I, the number of states is less than fifteen. It is thought that this is due to the conventional notation for describing a functional specification (natural language or state transition matrix, etc.). Thus, the limitation on the number of states may be changed by using an FD. In our experience, when the number of states is over twenty, the FD cannot be readily grasped by human operators.

   The mean number of input conditions varies widely depending on the software. In our experience with cause-effect graphs, the number of conditions (input conditions, output data, and head states in an FD) is limited to thirty or forty. There is a problem in the decision table (which is induced to an FD to describe boolean functions) in that the combination of conditions increases exponentially in proportion to the number of input conditions. Thus, the number of input conditions at each state should be kept as small as possible.
2. *Remarks on writing an FD.* It is noticeably easier to describe the ordered logic of a functional specification in an FD than in a cause-effect graph. However, when states are made without giving them much thought, nonexecutable test cases are apt to be generated by AGENT. Nonexecutable test cases are caused by inadequate attention to state setting. Such cases can be solved by state decomposition.
3. *Effective use of the AGENT method.* The AGENT method is particularly effective because it allows detection of design errors in a functional specification by translating it to an FD. Even inexperienced people are able to generate test cases with AGENT. The following factors enhance the effectiveness of the AGENT program.
   (a) *Description of an FD at the design stage.* This is effective for early detection of design errors. The FD review ensures that development people will have a common understanding of a functional specification.
   (b) *Use of inexperienced people for generation and execution of basic test cases in functional testing.* This allows experienced people to concentrate on cases not covered by the AGENT method or on very special cases which are deduced by experience and human intuition.

SUMMARY

The AGENT method has been proposed for systematic test case generation for functional testing. Generated test cases satisfy the following criteria for a function diagram which expresses a functional specification:

1. Each state includes true and false cases of input and output conditions.
2. All transitions of a state transition are retrieved at least once, and bypassing and getting through all loops in a state transition are included.

The AGENT program automatically generates test cases from a function diagram. While further evaluation is required, experience with the AGENT method and program have shown it to be effective for standardizing test case generation and simplifying test case management.

In future work, it will be necessary to evaluate our testing criteria by using error data gathered in the field. In order to widely propagate the AGENT method, much related research is needed. The first task is to extend the function diagram. For example, a function diagram should be able to describe parallel processing. The next task is to strengthen the testing criteria. To accomplish more effective testing, the AGENT method must be capable of generating not only test cases, but also test data for executing a program.

REFERENCES

1. Myers, G. J. *Software Reliability,* John Wiley & Sons, Inc., 1976.
2. ———, *The Art of Software Testing,* John Wiley & Sons, Inc., 1979.
3. Furukawa, Z., et al. "AGENT: Automatic Generation of Test-cases with Cause-Effect Graphs," *Proceedings of the 6th ICSE Poster Session,* 1982.
4. Howden, W. E. "Reliability of Path Analysis Testing Strategy." *IEEE Transaction on Software Engineering,* (Vol. SE 2-3), 1976.
5. Miller, E. F., Jr. "Coverage Measure Definitions Reviewed." *Testing Techniques Newsletter,* (Vol. 3-4), 1980.
6. Lanzarone, G. A. "Automatic Functional Test Case Generation for Realtime Control Systems." *Proceedings of the 6th ICSE Poster Session,* 1982.

A proposed causative software error classification scheme

by JAMES S. COLLOFELLO
Arizona State University
Tempe, Arizona

and

L. BLUMER BALCOM
GTE Sylvania
Mountain View, California

ABSTRACT

Various tools, techniques, and methodologies have been developed by software engineers over the last 15 years. A goal of many of these approaches is to increase product reliability and reduce its cost by decreasing the number and severity of errors introduced by the software development process. The collection of software error data would appear to be a natural means for validation of these software engineering techniques. Yet, current software error collection efforts have had limited success in this area. A new causative software error classification scheme is introduced in this paper to refine these data collection efforts so that they can be better used in software engineering validation studies.

INTRODUCTION

The major expenses in most computer systems can be attributed to their software components. Software development is a very complex process, usually resulting in a multitude of errors. The detection and removal of these software errors often consumes over half of all software development resources. In an effort to learn more about the software errors being introduced into their products, many organizations are attempting to collect and maintain software error data. These software error data collection efforts can provide needed information about the software development process.

One major application of these software error data is evaluating various tools, techniques, and methodologies developed by software engineers over the last 15 years. Since a major purpose of many of these tools and techniques is to increase product reliability and reduce its cost by decreasing the number and severity of errors, software error data collection provides a natural means for validation of these claims.

In addition to evaluating software engineering tools and techniques, error data may assist in the evaluation of reliability and productivity metrics. These data may also be useful in guiding project managers in resource allocation if certain types of errors are shown to occur more frequently in certain types of software units, or if certain tools are shown to reduce the number or severity of errors likely to occur in a project. Finally, error data may help guide the development of new tools by indicating weak areas.

In this paper the results of an extensive literature review of current software data collection schemes will be summarized. The weaknesses and inabilities of these schemes to satisfy all the potential benefits of data collection will be noted. A new causative software error classification scheme will then be proposed to overcome these deficiencies.

REVIEW OF ERROR CLASSIFICATION SCHEMES

In order to develop a useful error classification scheme, a comprehensive literature review of existing error classification schemes was performed. Although details of this study are contained in another publication,[1] a summary is given here. Errors have been classified by symptom, by cause, by the type of code in which a fault was found or a change was made, by the project phase in which the error was introduced, by severity, or (usually) by some combination of these.

A symptomatic error classification attempts to form groups describing the faults in a document or piece of code. These categories may be broad (e.g., logic, data handling, interface),[2] or specific (e.g., incorrect testing of a loop condition, uninitialized variable).[3] The data for this type of scheme are relatively easy to collect from a typical problem report, that is, a document describing some problem experienced with the software. Such a scheme is most useful in measuring the relative effectiveness of different error detection techniques.[3] The major problem with a system as detailed as Endres' is that many of the categories are project-specific and would disappear with a different type of project, set of hardware, and source language.

Causative error categories attempt to identify what should have been done differently in order to prevent the error. This sort of classification is more useful in evaluating software engineering tools, techniques, and measures. Endres has shown how a causative scheme may be derived from a symptomatic scheme. The same sort of technique was used in the TRW Software Reliability Study.[4] It must be noted, however, that this mapping is neither one-to-one nor onto. The common source of error data when a causative classification is used is the change report, a configuration control document, completed by the person who figured out what caused a particular fault and how to fix it.[4,5,6] Change reports typically indicate the change, but not the source of the problem. One problem with using the change report to make inferences about the source of an error, however, is that the change selected may have been chosen, not because it fixed the error, but because it was the easiest way around the problem.[6] This is likely to occur, for example, when requirements ambiguities or high-level design omissions are not discovered until late in development.

Errors are often classified by the type of code in which a fault or change occurs. This can provide useful feedback to programmers and testers for the purpose of concentrating their efforts.

Classification of errors by the project phase in which the error was committed is natural because the different activities associated with each phase are bound to introduce different types of errors. Thus, almost every error data collection scheme takes into account project phase.

Another type of classification scheme is based on the severity of the error, severity being defined by time, cost to fix, or impact on the system. Error severity, when measured by the impact the error has on the system, is usually divided into three categories—serious, major, and minor—where a serious error caused a system crash or prevented the accomplishment of a primary system objective, a major error noticeably reduced system performance, and a minor error was transparent to the user.[7]

To summarize, errors have been classified by symptom, by cause, by location of fault or change, by source development phase, and by severity. Most of the error data collection schemes involve a combination of two or more of these five types of schemes. All these categorizations can provide useful information for programmers, testers, managers, and researchers if the data are reliable. For the data to be reliable, errors must be classified by the person who has analyzed the problem and fixed the error. Because of this, the classification scheme must be easy to understand and use. The information of primary importance to researchers is that of error cause, which is also the most difficult to collect. For conclusions to be drawn about software in general, data must be collected from a variety of real-world software development environments.

PROPOSED ERROR CLASSIFICATION SCHEME

Analysis of previous software error data collection efforts and discussion with software quality assurance personnel in several companies yield some principles for the design of a new error data collection effort. (1) Data collection in the real world must be minimally obtrusive into a company's normal procedures in order to minimize expense and the effects of data collection on the data, and to maximize the reliability of the data. There are two principal effects of the data collection process on data: the time spent collecting the data detracting from the programming effort, and the Hawthorne effect—that is, the knowledge that a particular aspect of a task is being monitored affecting how it is performed. (2) It is desirable to collect as many pieces of information as possible to allow analysis of controlling factors, find unexpected results, and provide for future use of the data in as yet unformulated experiments. It is good scientific practice to record anything that could conceivably have an effect on the results of an experiment. This is especially true when studying complex human behavior and products. The expense of software development and maintenance mandates data collection, and the expense of the data collection[8] mandates efficient use of the data. (3) The data must cover as many different environments and projects as possible to validate any claims about software in general. This is true as long as we are unaware of many controlling factors and cannot control many of the ones we are aware of. (4) The effort must maximize both long- and short-term benefits for the data collectors, their managers, their companies. Data should be collected into a database that will help them concentrate their efforts on problem-prone areas of the current project and know the likely effect of various choices on software error content. (5) The new data collection effort should complement previous efforts to verify or refute their results for different environments, to allow use of their data in conjunction with the new in other experiments, and to obtain types of data that their effort failed to collect. In the remainder of this section a Software Change Report form designed with these principles in mind will be described.

A cursory examination of the Software Change Report form in the appendix reveals some important features of its design. The report differs from typical change reports in that it is designed specifically to deal with changes due to discovered mistakes. Changes due to newly requested capabilities, the lack of which cannot be attributed to the software development organization, should not be considered in software reliability studies or other error analyses. As with other change reports, it is intended to be completed by the person who diagnosed the problem and prescribed the cure, since this person will have the most insight into the type of information required. To make data collection as painless as possible for this often harried person, all responses are simple checkmarks and codable short answers. The elimination of prose responses will make data collection quicker and more reliable and analysis less expensive. It also permits data collection to be totally automated, with interactive error reports connected directly to the database.

This form is derived from the change report form developed by Ostrand and Weyuker,[9] who derived theirs from Weiss.[6] It differs mainly in two respects: (1) wording was neutralized so as to reflect as nearly as possible a consensus of current software development practices, rather than favoring any one company; (2) causative and symptomatic error classifications replace the prose responses, thus minimizing the time required to collect the data without biasing it with a researcher's interpretation for classification.

The approach taken here differs from that of Basili and Weiss[10] in that the report form was designed to handle all phases of the software life cycle. Providing a single form for use during all stages of development and maintenance will simplify the data collection process and reduce the time required to train personnel in its use. Though it is expected that the report form will evolve somewhat with changing research needs and refinements of the classification schemes, the constant use of a single, general-purpose form will essentially eliminate the Hawthorne effect after an initial startup period. Nevertheless, on the advice of Weiss,[6] each item on the change report was selected after brainstorming for research questions and error-related data needed to deal with them.

The first part of the Software Change Report contains configuration management data, error severity data, problem detection and isolation methods used, problem detection and isolation time estimates, a symptomatic error classification, and change properties of the error. This information is not new and is described in more detail in a technical report.[1] In the remainder of this section, emphasis will be placed on the causative nature of the error classification scheme.

The causative error classification is a two-dimensional scheme accounting for both the software life cycle phase during which the error originated and the behavior or conditions that caused the error. One problem with most software error classifiction schemes is that they fail to recognize that different kinds of errors are made in different phases of the software life cycle. This scheme was derived by noting the kinds of activities occurring and documentation produced in each phase of the life cycle, and then analyzing the kinds of errors that could be made, given the activities.

Recognizing that different companies divide the software life cycle differently, and that they organize the information produced at each stage into different types of documents, the first step in developing the two-dimensional causative error classification scheme was to define life cycle phases by the kinds of information produced. The intention here is not to dictate to a company how it should organize its software development and maintenance processes. Rather, the intention is to allow each software organization to identify which activities that they are already performing belong in the phases of the classification scheme. This way, meaningful comparisons of data between companies can take place without disrupting their operation.

For the purposes of this error data collection scheme, then, the software life cycle phases are defined by a set of activities and the resulting documentation. The Requirements Specification stage involves the communication and definition of functional and performance requirements. The documents produced in this stage may be thought of as a contract specifying what a system is supposed to do from the point of view of the designers, the testers, the customer, and the users.

The High Level Design phase involves the analysis of requirements and definition of a solution that will meet these requirements. The resulting documentation specifies the components of the system that will perform each required function and define how the various components are supposed to interact.

Detailed Design and Coding involve the implementation of component specifications with machine-understandable algorithms and data specifications. The resulting documents include, at the very least, compiled source listings, and often, explanatory prologue and higher-level algorithms in pseudocode, PDL, or flow chart form. Although low-level design is often considered a separate task from coding, both activities are characterized by the successive refinement of algorithms, and the types of errors possible are essentially the same. Errors made at this stage are mostly logical and arithmetic in nature.

One stage of software development appears to be missing from this scheme: testing and verification. The reason for this is that this error classification scheme is based on types of documentation produced, rather than a distinct period of time. Errors in the software that originate while a programmer is doing informal unit testing should be included in Low Level Design and Coding, and errors that occur after a component has been submitted for integration should be included under Debugging and Maintenance.

The final category in this dimension refers to additional errors which, because of the availability of preexisting documentation covering all development phases, may occur during Debugging and Maintenance. While all activities falling after the initial release of a software product have traditionally been lumped into a single category called Maintenance, these activities are usually similar to those of initial development. As in initial development, the requirements for a change and the environment must be understood, a design must be developed and implemented, the system must be retested, and documentation must be written to explain the change within the context of the system. The main difference is that there is some preexisting documentation, which may be used and must be maintained if it is to be used in the future.

The second dimension of this scheme consists of causative error categories for each phase. These are sorted into three rough groups corresponding to steps in the problem-solving process: (1) Understanding the task in all its dimensions: Communicational difficulties will cause errors of misunderstanding. (2) Analyzing the problem and synthesizing a solution: Conceptual difficulties will cause errors at this stage. (3) Documenting the solution: If the solution is going to be used by other people, it must be documented in such a way that the whole solution is clear. Clerical errors are the result of oversights in the process of documentation. Often, tools are devised to assist in one or another of these areas; therefore, while the categorization of errors at that level is rough, a concentration or dearth of errors in one of these rows may indicate the need for better tools to eliminate those kinds of errors or the success of a particular tool.

The error categories are described in detail with examples in another report.[1] A brief explanation of each error category is presented in Figure 1.

Requirements Specifications

Specification of requirements for function and performance in a new system or in a change to an existing system.

Communicational

Inadequate understanding of programming environment

The programming environment includes personnel, machinery, management, development tools, and any other factors which affect the programmer's ability to implement the requirements.

Inadequate understanding of implementation environment

The implementation environment includes the configuration of the system the new software is to replace, if any, and any interfacing machinery, software, personnel, or procedures.

Inadequate understanding of customer's requirements

Totally missing requirements are caused by an incomplete understanding of the customer's needs. The customer may not think to mention every detail because he figures that the professional knows what details are relevant, and that he will ask appropriate questions.

Miscommunication of capabilities to customer

Failure to explain the capabilities of a computing system may lead to an unsatisfactory product, with the subsequent need for enhancements.

Inadequate communication between team members

Confusion between team members regarding responsibilities or decisions.

Conceptual

Unnecessary requirement

Specification of a requirement the customer did not ask for.

Untestable requirement

Specification of a requirement whose fulfillment cannot be verified.

Figure 1—Error categories

Inconsistent requirements

Specification of conflicting requirements.

Unimplementable requirement

Specification of a requirement which cannot be fulfilled, given the state of the art and the system configuration.

Undocumented requirement

Omission of a requirement from one of the documents.

Clerical

Nonadherence to standards

"Standards" refers to standards of documentation enforced in a particular software house.

Other

Typographical and other miscellaneous errors.

High Level Design

Specification of system components to fulfill requirements. System design.

Communicational

Inadequate understanding of programming environment

Same as above.

Inadequate understanding of implementation environment

Same as above.

Misunderstanding of requirements

Inability of designer to understand a (set of) requirement(s), not traceable to a specific flaw in a Requirements Specifications document.

Inadequate communication between team members

Same as above.

Conceptual

Missing feature

The omission of a required feature from the system design, where a feature is a logically distinct part of the specification.

Unnecessary feature

The addition of a feature not required by the specifications.

Uncoordinated modules (usage of shared data, timing)

Failure of a designer to account for all interactions between components. This includes both specified and hidden interfaces.

Duplicated feature

Unnecessary duplication of a specified feature.

Insufficient capacity of design

System architecture which cannot meet a performance requirement when implemented.

Clerical

Vague specification

Not enough detail employed in the specification of a component.

Nonmatching interface specifications

Number or type of passed elements not matching.

Undocumented assumption

This category is for errors caused by designers making conflicting assumptions about interacting features because they were not documented.

Nonadherence to standards

Same as above.

Other

Same as above.

Detailed Design and Coding

Refinement of the design for each component and actual implementation.

Communicational

Misunderstanding of module or data definition

Inability of programmer to understand a higher level component specification, not traceable to a specific flaw in the higher level document.

Programmer unaware of convention

A convention might be stated once in a document (or orally) to avoid repeating it many times.

Conceptual

Unspecified feature added

The addition of a feature not required by the high level design.

Specified feature missing

The omission of a feature clearly specified in the high level design.

Unnecessary logic

Extra logic which reduces performance and does not improve comprehensibility of source code.

Missing, incorrect, or inconsistent logic

A logic sequence which does not correctly implement a feature.

Figure 1 (continued)

The following subcategories of incorrect logic should be used when applicable.

Timing error in concurrent modules

Flow control which results in deadlock, problems in accessing a critical section, or problems in the use of information deposited in a critical section.

Imprecise numerical approximation

Erroneous numerical analysis technique.

Missing exception handling (robustness)

Failure to provide for an exception condition.

Insufficient capacity of machine implementation

Module implementation which does not meet a performance requirement.

Clerical

Data definition not conforming to specification

Inaccurate transcription of data specification into implementation language.

Interface not conforming to specification

Inaccurate transcription of interface specification into implementation language.

Uneducated use of programming language features

Erroneous or poor implementation due to the programmer not taking advantage of a programming language feature.

Nonadherence to standards

Same as above.

Undocumented reasoning for difficult or unusual solution

Neglected documentation may cause problems when referencing or changing the source code.

Undocumented assumption

Same as above.

Other

Same as above.

Additional Errors incurred by altering existing products

During debugging and maintenance, these errors may occur because of the possible existence of all levels of documentation.

Communicational

Existing documentation inconsistent

Error caused by disagreement between two or more pieces of documentation.

Existing documentation incomplete

Error caused by a lack of information about requirements or earlier design decisions.

Misunderstanding of documentation

Inability of programmer to understand existing documentation, not traceable to a specific flaw therein.

Misinterpretation of language constructs

Failure to understand source code due to lack of familiarity with the programming language.

Conceptual

No categories.

Clerical

Revisions not made to all documentation

Failure to update all documentation.

Figure 1 (continued)

When classifying an error, the data collector should attempt to trace the error back to its source, checking only one category. If an error, for whatever reason, cannot be traced to an earlier stage of development, even though the problem seems to be a misunderstanding of something that has gone on before, the error should be classified in one of the communicational categories, which indicates a misunderstanding of the previous level.

CONCLUSION AND FUTURE RESEARCH

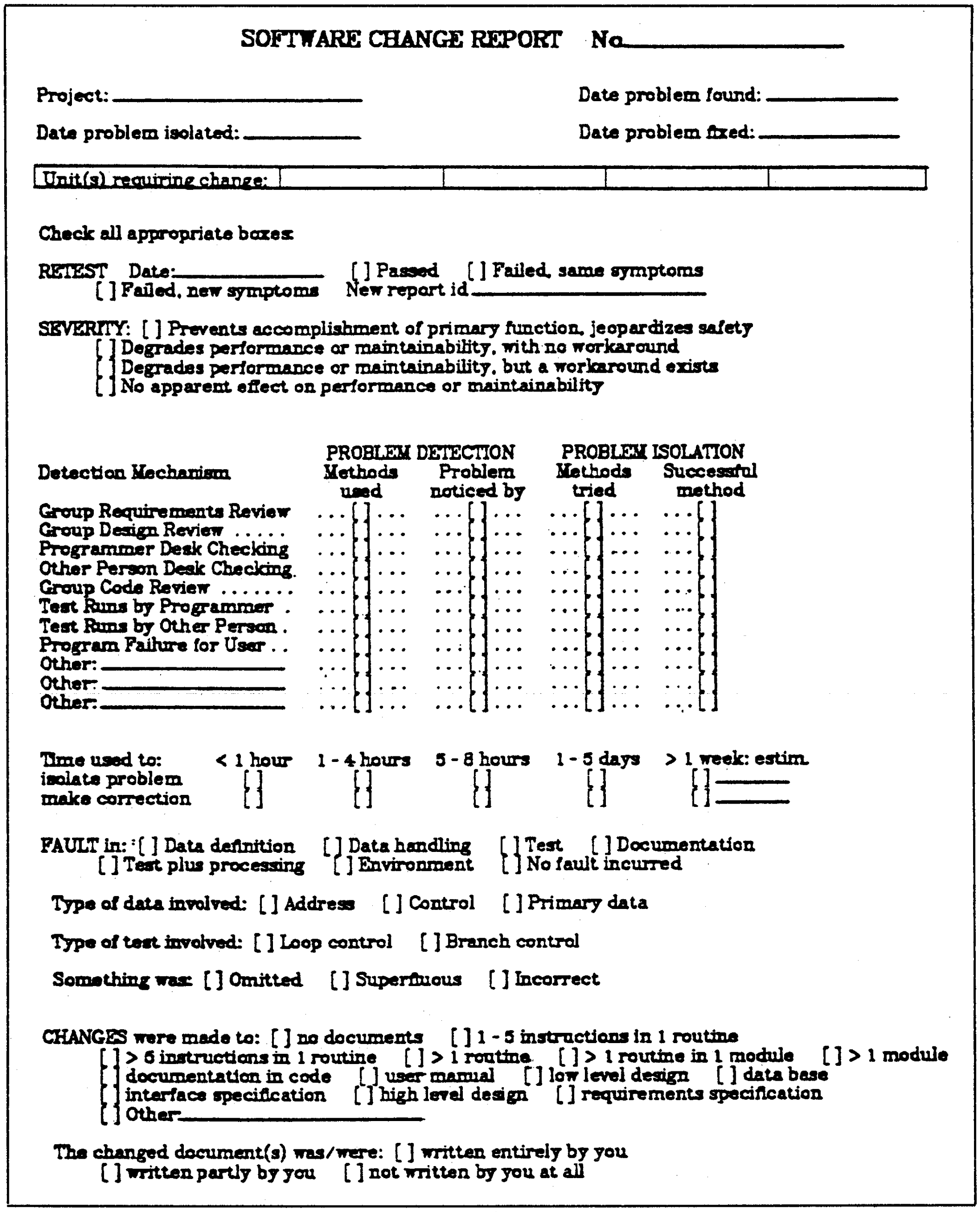
In this paper a causative software error classification scheme has been presented. This scheme differs from previous software error classification schemes by emphasizing the entire software life cycle and the causative attributes of the errors. Data carefully collected according to this scheme can be used to validate the claims of many software engineering tools and techniques.

Some limited experimentation with this error scheme has shown that classification of an error by life cycle phase and major causative categories is not difficult. More detailed classification within a single major category does, however, require more training. We plan to continue refining and experimenting with the causative error classification scheme. We also plan to attempt to calculate some measure of reliability for the error classification scheme that reflects its ability to capture accurate data. We will then attempt to use this measure to compare the reliability of existing schemes. This comparison must also take into account the differences in resolution of the different techniques.

REFERENCES

1. Collofello, J. S., and L. B. Blumer. *A General Scheme for Software Error Data Collection.* Arizona State University Computer Science Technical Report, June 1983.
2. Lipow, M. "Prediction of Software Failures." *Journal of System Software,* 1 (1979), pp. 71–75.
3. Endres, A. "An Analysis of Errors and Their Causes in System Programs." *IEEE Transactions in Software Engineering,* SE-1 (1975), pp. 140–149.
4. Thayer, T. A., M. Lipow, and E. C. Nelson. *Software Reliability: A Study of Large Project Reality.* Amsterdam: North-Holland, 1978.
5. Turner, C., G. Caron, and G. Brement. *DACS Data Compendium Series: NASA/SEL Development Data.* Rome, N.Y.: IIT Research Institute, 1982.
6. Weiss, D. M. "Evaluating Software Development by Error Analysis: The Data from the Architecture Research Facility." *Journal of System Software,* 12 (1975), pp. 57–70.
7. Bowen, J. B. "Standard Error Classification to Support Software Reliability." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 697–705.
8. McGarry, F. E. "What Have We Learned in the Last 6 Years: Measuring Software Development Technology." *Proceedings of the 7th Annual Software Engineering Workshop.* Greenbelt, Md.: Goddard Space Flight Center, No. SEL-82-007, 1982.
9. Ostrand, T. J., and E. J. Weyuker. "Collecting and Categorizing Software Error Data in an Industrial Environment." Technical Report 47, New York University, 1982.
10. Basili, V. R., and D. M. Weiss. "Evaluation of a Software Requirements Document by Analysis of Change Data." *Proceedings of the 5th International Conference on Software Engineering.* 1981.

APPENDIX



SOFTWARE CHANGE REPORT No.________________

Project: ________________ Date problem found: __________

Date problem isolated: __________ Date problem fixed: __________

Unit(s) requiring change:				

Check all appropriate boxes:

RETEST Date: __________ [ ] Passed [ ] Failed, same symptoms
[ ] Failed, new symptoms New report id __________

SEVERITY: [ ] Prevents accomplishment of primary function, jeopardizes safety
[ ] Degrades performance or maintainability, with no workaround
[ ] Degrades performance or maintainability, but a workaround exists
[ ] No apparent effect on performance or maintainability

	PROBLEM DETECTION		PROBLEM ISOLATION	
Detection Mechanism	Methods used	Problem noticed by	Methods tried	Successful method
Group Requirements Review	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Group Design Review	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Programmer Desk Checking	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Other Person Desk Checking.	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Group Code Review	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Test Runs by Programmer .	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Test Runs by Other Person .	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Program Failure for User . .	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Other: __________	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Other: __________	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]
Other: __________	... [ ] ...	... [ ] ...	... [ ] ...	... [ ]

Time used to:	< 1 hour	1 - 4 hours	5 - 8 hours	1 - 5 days	> 1 week: estim.
isolate problem	[ ]	[ ]	[ ]	[ ]	[ ] ______
make correction	[ ]	[ ]	[ ]	[ ]	[ ] ______

FAULT in: [ ] Data definition [ ] Data handling [ ] Test [ ] Documentation
[ ] Test plus processing [ ] Environment [ ] No fault incurred

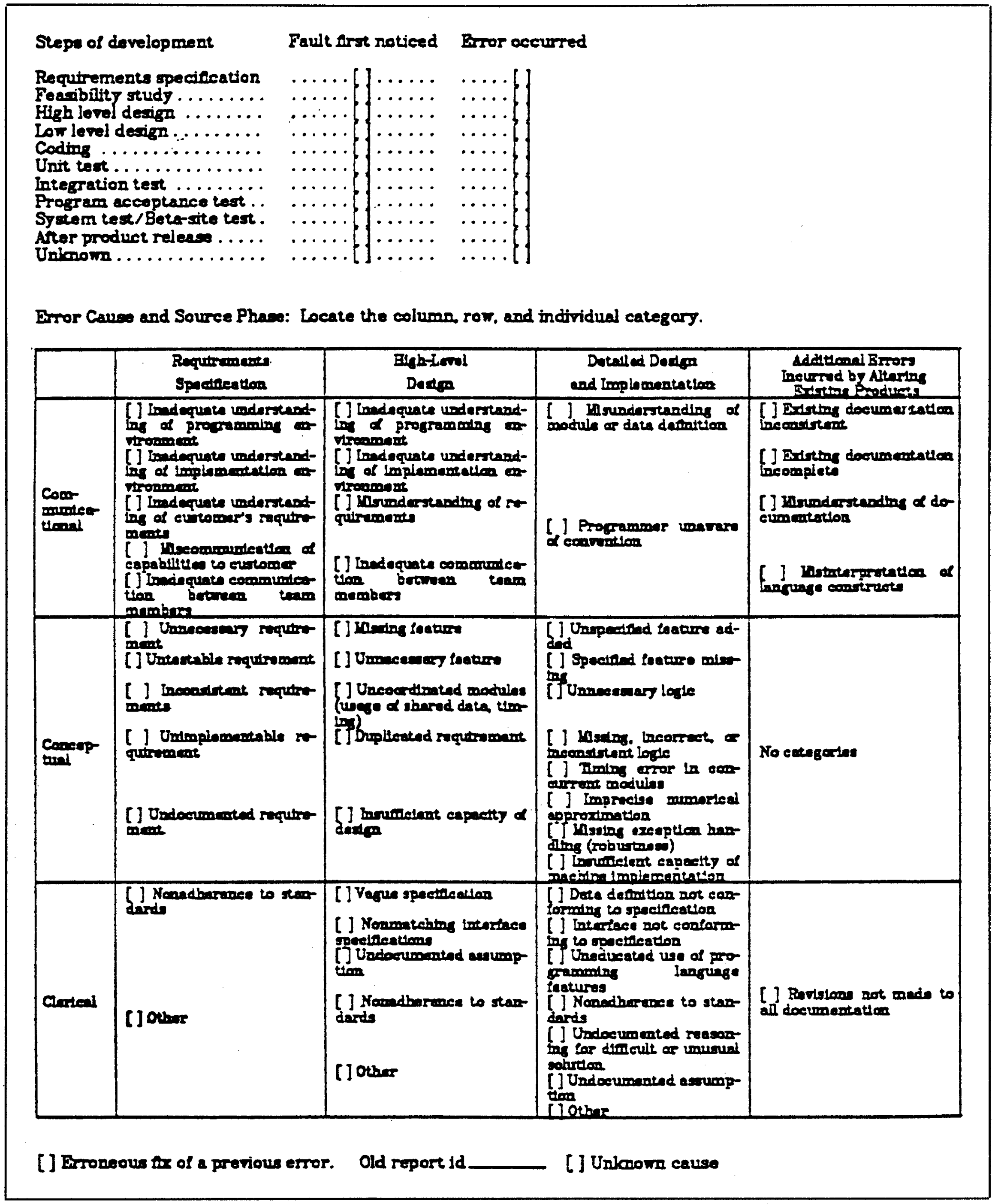
Type of data involved: [ ] Address [ ] Control [ ] Primary data

Type of test involved: [ ] Loop control [ ] Branch control

Something was: [ ] Omitted [ ] Superfluous [ ] Incorrect

CHANGES were made to: [ ] no documents [ ] 1 - 5 instructions in 1 routine
[ ] > 5 instructions in 1 routine [ ] > 1 routine [ ] > 1 routine in 1 module [ ] > 1 module
[ ] documentation in code [ ] user manual [ ] low level design [ ] data base
[ ] interface specification [ ] high level design [ ] requirements specification
[ ] Other ________________

The changed document(s) was/were: [ ] written entirely by you
[ ] written partly by you [ ] not written by you at all



Steps of development	Fault first noticed	Error occurred
Requirements specification	[ ]	[ ]
Feasibility study	[ ]	[ ]
High level design	[ ]	[ ]
Low level design	[ ]	[ ]
Coding	[ ]	[ ]
Unit test	[ ]	[ ]
Integration test	[ ]	[ ]
Program acceptance test	[ ]	[ ]
System test/Beta-site test	[ ]	[ ]
After product release	[ ]	[ ]
Unknown	[ ]	[ ]

Error Cause and Source Phase: Locate the column, row, and individual category.

	Requirements Specification	High-Level Design	Detailed Design and Implementation	Additional Errors Incurred by Altering Existing Products
Communicational	[ ] Inadequate understanding of programming environment [ ] Inadequate understanding of implementation environment [ ] Inadequate understanding of customer's requirements [ ] Miscommunication of capabilities to customer [ ] Inadequate communication between team members	[ ] Inadequate understanding of programming environment [ ] Inadequate understanding of implementation environment [ ] Misunderstanding of requirements [ ] Inadequate communication between team members	[ ] Misunderstanding of module or data definition [ ] Programmer unaware of convention	[ ] Existing documentation inconsistent [ ] Existing documentation incomplete [ ] Misunderstanding of documentation [ ] Misinterpretation of language constructs
Conceptual	[ ] Unnecessary requirement [ ] Untestable requirement [ ] Inconsistent requirements [ ] Unimplementable requirement [ ] Undocumented requirement	[ ] Missing feature [ ] Unnecessary feature [ ] Uncoordinated modules (usage of shared data, timing) [ ] Duplicated requirement [ ] Insufficient capacity of design	[ ] Unspecified feature added [ ] Specified feature missing [ ] Unnecessary logic [ ] Missing, incorrect, or inconsistent logic [ ] Timing error in concurrent modules [ ] Imprecise numerical approximation [ ] Missing exception handling (robustness) [ ] Insufficient capacity of machine implementation	No categories
Clerical	[ ] Nonadherence to standards [ ] Other	[ ] Vague specification [ ] Nonmatching interface specifications [ ] Undocumented assumption [ ] Nonadherence to standards [ ] Other	[ ] Data definition not conforming to specification [ ] Interface not conforming to specification [ ] Uneducated use of programming language features [ ] Nonadherence to standards [ ] Undocumented reasoning for difficult or unusual solution [ ] Undocumented assumption [ ] Other	[ ] Revisions not made to all documentation

[ ] Erroneous fix of a previous error. Old report id ________ [ ] Unknown cause

Syntactic information useful for software maintenance

by JAMES S. COLLOFELLO and JOHELEN W. BLAYLOCK
Arizona State University
Tempe, Arizona

ABSTRACT

Software is an expensive asset, one which costs more to maintain than to acquire. The high cost of maintenance has been attributed to low programmer productivity in maintenance. This is due in part to the lack of tools existing within an integrated environment which address the full spectrum of maintenance activities. This paper addresses these concerns through the discussion of an ambitious research project to develop a Maintenance Engineering Environment (ME2). ME2 consists of three components: (1) a knowledge base, (2) an integrated toolset, and (3) maintenance personnel. Following a general discussion of ME2, this paper presents research results that identify the set of syntactic information needed to perform software maintenance activities. This information is critical to establishing the foundation of the maintenance-knowledge base in the ME2 project. The results also demonstrate the feasibility of evaluating source code for maintenance-specific information.

INTRODUCTION

There are a number of problems associated with maintenance, the greatest of these being that a programmer must understand a system before modifying it. A simple solution to this problem would be to retain the development team to function as the maintenance staff, but this is not always possible. Due to high programmer turnover in the industry coupled with a stigma attached to the maintenance task,[1] it is difficult to retain experienced personnel in maintenance. Thus, maintenance is performed by unskilled programmers[2] who must put considerable effort into learning the workings of the system before they can competently modify it. This learning task is often obstructed by a number of problems; documentation may be non-existent, insufficient, or outdated and useless, leaving the programmer with only the code to go by; the code may be unstructured and highly complex, making it difficult to understand; and the code may have been subjected to numerous modifications by other programmers who *also* did not understand the program very well, thus degrading system structure and system robustness.[3]

It has been suggested that one way to improve maintenance productivity is to produce development tools that can be carried over into the maintenance phase,[4] but development tools do not always translate well to the maintenance task because they may not address a number of maintenance problems. One maintenance problem that does not exist during development is the task of coming to understand an unfamiliar program. This stems from the fact that personnel with intimate system knowledge acquired during design are largely unavailable for making changes during development. Another maintenance-specific problem is the necessity of working within established system constraints such as available variable storage space and performance requirements. In addition to the problems peculiar to maintenance, there are many programs in existence which were not developed with the aid of advanced tools that must be maintained. These factors coupled with the high cost of software maintenance point to the need for a maintenance environment directed at improving maintenance productivity by reducing the time spent by maintenance programmers to acquire knowledge about unfamiliar programs, design and install modifications, trace ripple effect, and retest the system.

A MAINTENANCE ENGINEERING ENVIRONMENT

A project is currently underway at Arizona State University to produce a prototype Maintenance Engineering Environment (ME2).[5] The environment will be applicable for both programs with no support documents other than the source code, and programs with a full complement of support information. ME2 has three components: (1) a maintenance knowledge base, the foundation of the maintenance environment, (2) an integrated toolset, and (3) maintenance personnel.

The Maintenance-Knowledge Base

The maintenance-knowledge base is the core of ME2, providing the power of a data base through storage, modification, retrieval and deletion of data. As the maintenance personnel interact with the information, the knowledge base "learns" by deriving new information from the interaction and adding it to the environment. With the new information, the knowledge base is able to create new relationships not explicitly given to it by the maintenance personnel.

The Integrated Toolset

The ME2 toolset focuses on technical support tools covering each of the maintenance activities of understanding, modification, and ripple effect analysis.

The tools directed at the first maintenance task, that of aiding a programmer to understand a program, provide the user with a high-level, problem-knowledge domain view of the program by providing program requirements information. At the lower-level program-knowledge domain, the maintenance programmer is able to view a graphic display of the module-calling hierarchy, along with module imports and exports for each module. Syntactic information on module import and export variables and semantic information on the module and its purpose is also available.

During the second maintenance task, that of modifying the software, ME2 provides high traceability from requirements through design of the code. ME2 also supports software change control management.

A ripple effect[6] analyzer tool is incorporated into the maintenance environment to support the third maintenance activity. For each module in the program, assumptions are identified which must be satisfied for correct operation of the module. The module, in turn, makes "decisions" based upon these assumptions. These decisions, in turn, may affect other assumptions. When a decision in a module is changed during maintenance, the change is statically traced to determine if the assumptions dependent upon the decision have been affected.

Cost-effective regression testing, although very difficult, is also under investigation for the maintenance environment.

The Maintenance Personnel

As noted, experienced maintenance personnel are difficult to acquire and keep. They are also the most valuable element in the maintenance environment. ME2 proposes to capture experience gained in the maintenance-knowledge base for the purpose of simplifying the learning task for replacement personnel. ME2 accomplishes this by querying the programmer for semantic problem and program knowledge acquired during the maintenance process. Thus, as more maintenance experience is acquired and captured, the power of the ME2 toolset is increased.

SYNTACTIC INFORMATION USEFUL TO MAINTENANCE

A key aspect of the ME2 project is defining information which will be useful in the maintenance task. This information is the substance of the maintenance-knowledge base and is based upon a foundation of syntactic information derived from the program. In order to describe the set of syntactic information which facilitates maintenance, a set of templates was developed. These templates were created to be language independent. This means that some fields in the template may have information which is syntactically available from some languages but only semantically available from others. The set of templates for capturing syntactic information are provided at the end of this paper, and are described below.

Control Flow Information

Knowledge of a program's control flow is a key component in gaining an overall understanding of how a program works, and in ripple effect analysis and regression testing. Control flow information is also basic to an understanding of program data flow.

Preliminary to the task of understanding a program's control flow is the task of identifying program *basic blocks* and modules. Basic blocks are defined as a "maximal group of statements such that no transfer occurs into a group except to the first statement in that group, and once the first statement is executed, all statements in the group are executed sequentially."[7] Modules are defined as the subprogram facility for the particular language. In Pascal, modules are the defined procedures and functions; in Fortran, they are subroutines and functions. Assembly languages may provide module facilities through a Jump-to-subroutine (JSR) or some other type of command which may use a stack to invoke and return from a subprogram.

Once the set of program blocks and modules is defined, inter-block control flow can be modeled using a directed graph where each node of the graph corresponds to a basic block. There is an arc (x, y) from block x to block y if control can potentially transfer from block x to block y at run-time.[7] Intermodule control flow may be captured in a representation of the module-calling network of the program using a similar graphing technique. Both inter-block and inter-module control flow information may be derived from a syntactic analysis of the program.

Control flow information is useful to the maintenance task in a number of ways; identification of program blocks, modules, and processes is basic to identification of the control flow between these entities; block and module flow networks provide the programmer with information on how the program modules interact; and information used to create a reachability matrix for use during regression testing is available from inter-block control flow information gleaned from the program. Control flow is also a foundation upon which to build an understanding of program data flow.

Data Flow Information

Basic to a discussion of data flow analysis are the terms *definition* and *use.* A variable definition can modify the value of the variable, as by an assignment or a READ statement. A variable is used if it is referenced without modification, as by a WRITE statement or when it is the operand of a computation.

Three concepts of data flow analysis are built upon knowledge of program control flow and variable definitions and uses: (1) reaching definitions, (2) live variables, and (3) use-definition chains. The first concept, *reaching definitions,* a term used by Hecht,[7] describes the problem of determining the sets of variable definitions that can "reach" the top of each node in an inter-block control flow graph. The second concept, live variables, deals with determining the set of variables that may be used after control passes to a given point in the control flow graph. The third concept, use-definition chains, involves linking the definitions of variables to their uses in the definitions of other variables. Variable uses are also linked backwards to the definition of that variable. This double linking from definition to uses and use to definitions forms a chain of data flow information which may be traversed for flow analysis.

Information which is basic to each of these data flow concepts is derived from knowledge of variable definitions and uses. The use and definition information may be derived from a syntactic analysis of program source code, allowing the sets of reaching definitions, live variables, and use-definition chains which describe inter-block data flow to be generated automatically.

In languages which provide a facility for modules, global data flow describes the inter-module flow of information through parameters and global data structures. When analyzing a program to determine inter-module data flow, it is necessary to capture the set of global data structures and parameters as well as their mode of passing.

Both inter-block and inter-module data flow analysis provide the maintenance programmer with knowledge of the flow of information through a program. Such knowledge is indispensable when attempting to understand how a program works. Data flow information also facilitates the tracking of program errors by providing the information flow which can be followed to find the location where program data goes bad.

Logical ripple effect tracing may be supported using use-definition chains to identify assumptions on variable values. This information along with inter-block control flow may be used to identify the set of program blocks which must be examined for ripple effect problems for each potential modification.

Declaration Information

Declaration information is the lowest available level of syntactic information above that available from the source code itself. It includes the variable's name and type. If the language provides a facility for declaring the variable's valid range of values, initial value, or specific usage (i.e., in COBOL, the programmer is allowed to declare the variable as either computational or display), this information should also be recorded to provide the maintainer with information on the structure and initial status of the variable. Variable aliases should be listed in the declaration information in order to determine correct data flow information.

The variable's defining module should also be noted. This, along with scoping rules for the language (possibly recorded as the list of modules that the variable is visible within) will provide the maintenance programmer with information on where the variable may be modified throughout the program.

Module declaration information includes the module name, type (if defined), and parameters. For parameters, information on name, type, and method of passing is also needed. A full set of declaration information for all objects defined within the module should include variables, other modules, constants user-defined types, labels, and external files. This information provides the maintenance programmer with an understanding of the environment created by the module.

Information on *module visibility* is also included in the declaration information. This entails not only what modules are visible to the given module, but what other modules can "see" and therefore invoke the given module. This information is the basis for defining the module-declaring hierarchy.

Constant names and their corresponding *values* are basic to the declaration information to be recorded for constants. Identification of the module that the constant is declared in provides the programmer with information on the visibility of the constant and which modules may therefore reference the constant. It is also necessary to capture the locations of references to the constant to provide the maintainer with information on what areas of the program will be affected if the constant is modified.

Information on *types* provides the maintainer with a set of templates for identifying variable structures. Information on user-defined types should include the name of the type and the definition of the type, including complete definitions of all subfields and indexes.

Label name and *location* form the basis of label declaration information. The visibility of the label should be included to provide the programmer with a reference on what program areas may jump to the label. Information on jumps to the label is available from the control flow information.

Declaration information is useful to the programmer when trying to establish a knowledge of a program's naming conventions and data structures. Declaration information tells the programmer what the data structures used by the program look like and where they are used, but does not tell the programmer how the structures are used or what information they may contain at any time.

Every program contains a wealth of syntactic information. The template for cross reference information at the end of this paper does not attempt to identify all existing cross reference information, but focuses on that information which is particularly useful to the maintenance programmer for program support. It should be noted that the set of information available from any given program is language dependent.

THE ME2 SYNTACTIC ANALYZER

To demonstrate the utility of a maintenance environment, a prototype is currently under development at Arizona State University. As part of this environment, a syntactic analyzer has been developed to analyze Pascal source code for information useful to the maintenance task. The analyzer accepts Pascal source code that has already been verified by a compiler for syntactic correctness, and outputs information to the knowledge base. The information produced is similar to that represented in the templates. The user may then interact with the knowledge base to access information for the purpose of understanding or modifying a program, or analyzing a program for potential ripple effect problems.

CONCLUSIONS AND FUTURE RESEARCH

The prototype syntactic analyzer implemented as part of the maintenance environment currently under development at Arizona State University demonstrates the feasibility of evaluating source code for maintenance-specific information. The utility of this information is clear in view of the need for consistent, correct, and current program information for support of the maintenance tasks of program understanding, program modification, and analyzing a given modification for potential ripple effect. The implementation of this analyzer has shown that performance of the maintenance task may be greatly enhanced at relatively low cost by providing the programmer with an automated means for acquiring information necessary to perform the maintenance task.

Our future research efforts are centered upon developing other aspects of the maintenance environment including expansion of the knowledge base to contain semantic information and the building of a powerful grapic user interface.

MAINTENANCE INFORMATION TEMPLATES

Control Flow Template

1. Identification of blocks and modules
   a. *Basic block identification.* Basic blocks are sequences of consecutive instructions that are always executed from start to finish.
   b. *Module identification.* Modules are characterized as the subroutine facility provided by the language. If no subroutine facility is available, only program blocks may be identified.
   c. *Process identification.* Processes are units of independently executing code. They may be identified by defining requestor and server modules within the program.

2. Identification of inter-block control flow. This information may be shown using the graphing technique described by M. S. Hecht.[7]
3. Identification of inter-module control flow. This information may be captured in a module-calling network. Recursive modules and the existence of abstractions, macros or library routines in the network should be noted, along with their interfaces.
4. Identification of inter-process control flow. This includes identifying corresponding requestor and server tasks. Exact rendezvous sequencing information may not be derived from program syntax since it is determined at run time.

Data Flow Template

Note: completion of the control flow template for a program is a prerequisite to identification of data flow.

1. Basic data flow identification
   a. *Variable definitions.* These are locations where the variable may be modified.
   b. *Variable uses.* These are variable references without modification.
2. Inter-block data flow
   a. *Identification of the set of reaching definitions.* These are variable definitions that can reach a given block due to the existence of definition clear-paths from the definition to the beginning of the block. This information may be collected by identifying all variable definitions within a block and then establishing whether the definitions reach the subsequent block by determining if there is a definition clear-path from the definition to the end of the block.
   b. *Identification of live variables.* These are variables which reach a given block and are also used within that block. This information may be collected by intersecting the set of reaching definitions with the set of variables used before being redefined by the given block.
   c. *Identification of use-definition chains.* These connect variable uses to definitions and vice-versa. This information is collected by associating every use of a variable with the most recent definition of that variable, and associating every definition of a variable with the uses of the variable that depend on that definition.
   d. *Identification of ASSERTs.* This includes identifying the assertion to be satisfied and its location in the program. This information, along with inter-block control flow, is used for ripple effect analysis.
3. Inter-module data flow
   a. *Identification of parameters and method of parameter passing.* This includes position of parameter in an argument list and is language dependent.
   b. *Identification of non-local, non-parameter variables (global variables) which are used by a module.*

Declarations Template

1. Variable Information
   a. *Variable name.*
   b. *Variable's defining module.* Include scoping information as it pertains to the variable.
   c. *Variable type.* If record-type, include subfield definitions; if array, include index types; for pointers, include access types.
   d. *Variable range of valid values* (for languages which provide this facility).
   e. *Variable initial value* (if available from definition).
   f. *Variable aliases.* For each alias indicate where in the program the alias is used.
   g. *Variable usage.* (i.e., COBOL classifies variables as either computational or display).
2. Module information
   a. *Module name.*
   b. *Module parent or defining module.* Include information on nesting and visibility.
   c. *Module type* (if defined).
   d. *Module parameters.* Name and position plus a full set of variable cross reference information for each parameter.
   e. *Module locally defined entities.* Full set of cross reference information for each variable, module, constant, type, file or label.
   f. *Generic instantiation.* If module is a generic instantiation, as provided by Ada, what is the generic instantiated and what types are given to the parameters?
   g. *Overloading.* Does module represent overloading of a system-defined function or operator?
   h. *Abstractions.* If module represents an abstraction, what are the defined operations on the abstraction?
3. Constant information
   a. *Constant name.*
   b. *Constant defining module.* Include scoping information as it pertains to the constant.
   c. *Constant value.* If constant is a deferred evaluation expression, note expression.
   d. *Reference locations.* Include all locations where a constant is used.
4. Overloaded operators
   a. *New name assigned to operator.* If a new function is assigned to the operator, detail it under *module information.*
5. User-defined types information
   a. *Type name.*
   b. *Type defining module.* Include scoping information.
   c. *Type definition.* Define types for all subfields and indexes. Note if type is a derived type. In Ada, a derived type represents a variable type with the same structure and name as an existing type, however, for the purposes of type checking, a derived type is considered unique.
   d. *Subtypes.* If type is a subtype indicate parent type.
6. Labels information
   a. *Label name.*
   b. *Label location.*
   c. *Label visibility information.*
7. External files information
   a. *File name.*
   b. *File type.* Full description of type needed.

c. *File used for input, output or both.* Include locations of opens and closes for each use.
d. *File access information (random or sequential).* If random, detail locations of file pointer manipulations.
e. *Device file is assigned to, if available.*
f. *File sentinel, if defined* (as in COBOL).

REFERENCES

1. Reutter, J., III. "Maintenance is a Management Problem and a Programmer's Opportunity." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 343–348.
2. Schwartz, B. "Eight Myths About Software Maintenance." *Datamation,* 28, 9 (Aug. 1982), pp. 125–128.
3. Mohanty, S. N. "Entropy Metrics for Software Design Evaluation." *The Journal of Systems and Software,* 2 (1981), pp. 39–46.
4. Yau, S. S. and Collofello, J. S. "Some Stability Measures for Software Maintenance." *IEEE Transactions. Software Engineering,* SE-6, 6 (Nov. 1980), pp. 545–552.
5. Collofello, J. S. and Woodfield, S. N. "A Proposed Software Maintenance Environment." *Proceedings of the IEEE Maintenance Workshop,* 1983, pp. 71–77.
6. Yau, S. S., Collofello, J. S. and MacGregor, T. "Ripple Effect Analysis of Software Maintenance." *Proceedings of IEEE COMPSAC 78,* Chicago, IL, Nov. 1978, pp. 60–65.
7. Hecht, M. S. *Flow Analysis of Computer Programs.* New York: Elsevier North-Holland, Inc., 1977.

Database performance optimization

by DALIA MOTZKIN
Western Michigan University
Kalamazoo, Michigan

ABSTRACT

A generalized model for the optimization of relational databases has been developed and implemented. The model, an extension of previous works, is more general and more complete than former models. It consists of a set of algorithms and cost equations, and its output is an optimal set of indices for all fields of all files in the database. It also determines which fields should not be indexed. It allows the user to indicate indices to be evaluated and it takes into consideration periodic reorganization, a variety of transaction types, and the multifield/multifile effects of some transaction types. It distinguishes between dense and nondense attributes, between primary and secondary fields, and between sorted and unsorted files. The optimal physical database configuration produced by the model is generated so that it can work within reasonable system constraints of time and space.

INTRODUCTION

The design of physical database is concerned with the optimization of access time and space requirements and with the prediction of database performance. These problems have been approached from two sides. One aspect is query optimization, and the other is the selection of an optimal set of indices and reorganization points. This paper concentrates on the second aspect.

The selection of the indices of a database is an important part of physical database design. Since index performances vary, there is a need to select the most suitable index for each field of each file. Whereas appropriate indexing improves performance considerably, excessive indexing can result in major performance degradation as well as in significant increases in storage requirements. performance of some types of indices deteriorates in time due to overflow situations and other problems. Reorganization is then required.

Problems of modeling, optimization, and prediction of database performance have been studied by many researchers, and interesting results have been published.[1–3,4–8,10–12,16,17,19,21,23] Additional bibliography, related to earlier work, can be found in extensive surveys.[18,20,22] However, previous modeling and optimization techniques are not complete; they suffer from one or more of the following problems:

1. The work is file-oriented rather than database-oriented.
2. The list of evaluated indices is inadequate: In some models only a few indices can be evaluated, having omitted the entire B-tree family and other indices. In some, a variety of updating techniques is not incorporated. Others compare too many indices rendering the models slow and inefficient.
3. Periodic reorganization is not addressed: Some file organizations and index structures require periodic reorganization due to overflow and other problems. Some models do not take reorganization into consideration in the selection process.
4. System constraints are not taken into consideration: Many computer systems, especially microcomputers, have limited amount of space. Other systems have time constraints due to heavy workload. These constraints do not play a role in many current models.
5. Important transaction types and the effects of these transactions are not included: Some models are oriented toward retrieval only. Others take into consideration maintenance operations such as insertions, deletions, and modification, but do not include the multi-index/multifield effects of transaction.

   The following example may illustrate a problem of the kind cited in 5: field i (say salary field) of record R has to be modified. To find the record R, the system uses the value of field j (say social security number). An index on field j will obviously improve the search performance, but an index on field i will not contribute to the speed of the search. On the other hand, the index on field i will have to be modified, thus decreasing the performance of this transaction.
6. The interaction and distinction between primary and secondary fields are lacking. Some models are oriented toward primary fields only, others toward secondary fields. Some models evaluate indices for all fields but do not distinguish between primary and secondary fields.
7. There is no discrimination between dense and nondense field attributes.
8. Performance prediction is not provided: Some models present optimal database configuration and/or reorganization points. But they do not provide the user with the time and space requirements of the database.

This work is an extension of previous work integrating the performance issues 1 to 8 above. The databases considered are assumed to be relational. All files are assumed to be in first normal form (possibly they are also in any or all other normal forms). No reference is made to modeling network or hierarchical databases.

DESCRIPTION OF THE MODEL

The model is composed of four parts: input parameters, the algorithm, performance and cost equations, and the output. We will first describe the input so that the parameters affecting the model will be evident; we will then describe the output (i.e., the outcome of the model). This will be followed by a general description of the algorithm, the computations, and the assumptions made. The detailed formulas used can be found in Motzkin.[15]

Input Parameters

The input to the system consists of four groups of parameters: system parameters, database parameters, user workload parameters, and index parameters. (For examples of parameters, see the section below, "Experimental Results."

System parameters

These parameters are concerned with system constraints and costs. They include total number of available blocks, the blocking factor (number of characters per block), average access time, cost per block (per day), and cost per access. Note that the total time available per unit of time, such as a day or a month, is not an input parameter. The time required to execute user workload is provided as part of the output. The user can modify the space allocation and achieve better access time. This procedure is described below in the section "An Overview of the Algorithms, Computations, and Assumptions."

Database parameters

These parameters provide database information such as the number of files, the number and names of fields in each file, and needed information on each field. The parameters include number of files; for each file, the name of the file, a flag indicating whether the file is sorted or unsorted, the name of the field on which the file is sorted, the name of the primary field and the total number of records in the file; for each field, the name of the field, a flag indicating whether the field attribute is dense or not dense (this parameter is needed because different types of indices are suitable for dense attributes and nondense attributes), the number of distinct attribute values (this is meaningful only for nondense attributes), and the number of characters in the field.

User workload parameters

This group of parameters is concerned with various transactions such as retrievals (also referred to as searches), insertions, deletions, modifications (also referred to as updates), and frequency of reorganization.

It is difficult to obtain the values for user workload parameters. The values may be estimated, or a program may count them over a period of time and come up with an average per unit of time such as a day or a month. This paper does not discuss methods by which user workload parameters may be obtained. It is assumed that such input is available for the model. User workload parameters include processes that can use indices; thus accesses that are results of operations such as PROJECT are not included.

The user workload parameters include total number of records inserted in each file per unit of time, total number of records deleted from each file per unit of time, total number of searches per field per unit of time, total number of updates per field per unit of time, and frequency of merge. We used the day as the unit of time. Note that some transactions are required for each file, whereas other transactions are required for each field.

Insertions and deletions are measured per file because when a record is inserted or deleted, all indices have to be modified. Searches are measured per field. Database users usually request records with given field values. Updates (modifications) also affect individual fields; for example, if a salary field is changed, only the salary index is modified. It is assumed that the searches for records to be deleted or modified are done using the primary field. The frequency-of-merge parameter indicates how often reorganization is done. Reorganization involves merging overflow areas with main areas, removing empty areas that might have been created as a result of deletions, regenerating some indices, and other related operations. This input parameter is required per file, since each file is reorganized along with the file indices.

Index parameters

The system provides a few of the more widely used indices as a default option. The user may add any additional indices to be used in the valuation.

Default option: The default option is used when the user does not specify her/his choice of indices.

For dense fields the system evaluates the B-tree index and sequential index. The B-tree index is chosen as a representative of the B-tree family, which includes B-tree, B$^+$tree, B*tree, and multilevel sequential index with block-splitting techniques used for updating. These four directories have similar performance; therefore, one representative is selected. The formulas for the B-tree are taken from Horowitz and Sahni.[9] The B-tree family has a very efficient access time, but space utilization may be as low as 50%. Therefore, the other directory chosen as a default option is a simple one-level sequential directory. The sequential directory is not as fast as a B-tree, but it is more economical than a B-tree in space requirements. In some database environments, especially on small computers, the space constraints may be stronger than the time constraints; thus the slower but more economical sequential directory may be more suitable.

For the nondense attribute an inverted file is selected as the default option. It was pointed out by Motzkin[14] and others that inverted files are superior to multilists in most situations. The uniform organization of inverted files[13] is assumed. The detailed formulas used can be found in Motzkin.[15]

Additional user-selected indices to be evaluated

A user may wish to evaluate and compare indices other than the default option ones. It is possible to enter additional indices and their characteristics. The system will incorporate all additional indices into the optimization process.

Output

The output includes total time required by the database operations described above per unit of time (per month in our implementation); total space required by the database; related cost of the database; and a list of files, fields, and selected indices—as well as fields for which not having an index was more cost effective. (For sample output see the section "Experimental Results.")

An Overview of the Algorithm, Computations, and Assumptions

For each field of each file the system first selects the best index (that with the lowest cost) out of all indices to be evaluated. In selecting the best index, the following cost considerations are included:

1. The cost of retrievals (searches) that use the index
2. The cost of index modifications due to insertions and deletions to the corresponding file
3. The cost of index modifications due to changes of corresponding field values in the corresponding file
4. The cost of space occupied by the index
5. The cost of index reorganization due to overflow and other deterioration factors

The searches for records to be deleted and modified are assumed to be done using the primary field. They are added to the cost of each index evaluated for each primary field of each file.

After the best index has been determined for a field, the cost of related processing of the field without an index is computed. Cost without an index will include the cost of direct search in the file for records associated with certain field values. Obviously the cost of direct search in the file will be significantly higher than the search that uses an index; however, there will be no cost of index space and index maintenance.

Now, for each field, the cost without index is compared with the cost with the best index. It is then determined whether the best index or no index will be selected for the field.

When indices (or no indices) are selected for all fields, the total database space is computed, including the space occupied by the files and the indices. If the total database space is greater than the available space, then the least useful index is removed. The process of removing the least useful index continues until enough indices have been removed yielding a total database space that is less than or equal to the available space (see Figure 1—Outline of the algorithm).

The usefulness of an index is determined by the difference between the cost associated with the corresponding field if an index is not used for the field and the cost associated with the field when an index (the best) is provided. An index is considered less useful if it does not reduce the cost of the field considerably. An index is normally more useful when the corresponding field has more searches and less modification, and if the index does not occupy a very large amount of space. (The exact formulas used in the cost equations can be found in Motzkin.[15])

At the end of the computations the user is presented with the total space, the total time of accesses computed from the output, and the related cost of the database. It is possible that

```
FOR i = 1 TO number of files DO
     FIND the best index for the primary field p; denote it by $IND_{i,p}$
     FIND whether it is "better" to have $IND_{i,p}$ or no index for field p of file i
     FOR j = 1 TO number of fields in file i DO
         IF I ≠ P
         THEN find the "best" index for field j; denote it by $IND_{i,j}$. Find whether it is better to have $IND_{i,j}$ or no index
              for field j of file i. Store the information on $IND_{i,j}$.
         ENDIF
    END FOR STATEMENT
END FOR STATEMENT
Compute total database space (include space requirement for files and indices).
    IF total database > total space available
    THEN
        FOR i = 1 TO number of files DO
            FOR j = 1 TO number of fields in file i DO
                $USEFUL_{i,j}$ = $COST\_OF\_FIELD_{i,j}$ without index − $COST\_OF\_FIELD_{i,j}$ with index
            END OF FOR STATEMENT
        END OF FOR STATEMENT
        SORT $USEFUL_{i,j}$ denote the sorted list $USEFUL^{k}_{i,j}$
                        (k = 1 for $USEFUL_{i,j}$ with smallest value and k = number of indices for $USEFUL_{i,j}$ with
                          highest value of USEFUL.)
    FOR k = 1 TO number of indices DO
        Remove $IND^{k}_{i,j}$ from database
        Database Space ← Database Space − Space of $IND^{k}_{i,j}$
        IF Database Space ≤ Available Space
        THEN Exit Loop
    END OF FOR STATEMENT
        END IF
        Compute Database Cost and Time
        Print output reports
```

Figure 1—Outline of the algorithm

while the space is acceptable, the time figure is too large. The database designer may then allow for more space for the database and run the optimization program again. The additional space allocation will allow the database to use more indices and thus improve the time figure. The user may also try to put less weight on the space by reducing the cost of a block; this reduction may also increase the number of indices used. The frequency-of-merge parameters can also be changed. This iterative procedure may continue until an acceptable configuration is achieved or until there is no further improvement.

Outline of the Algorithm

An outline of the algorithm appears in Figure 1.

A note on the complexity of the algorithm:

The separability assumptions have been used.[4, 19] Thus the computations are performed on each field of each file separately. Denote the total number of fields over all files of the data base by NF, and denote the number of indices to be evaluated by NI. Then the time required for the optimization process is $T = NF \cdot (NI + 1)$. Each additional iteration will take another T time.

EXPERIMENTAL RESULTS

The optimization and prediction model has been implemented by a PASCAL program. Four different simulation runs are provided (Figures 2–5). Field 1 is assumed to be the primary field in all files. The input parameter FILE TYPE with values U or S means unsorted or sorted file. Sorted files in this implementation are assumed to be sorted on the primary field. The FIELD TYPE parameter with values D or N means dense or nondense attributes. The time and cost figures are related to the accesses and maintenance parameters that were included in the input. (Processes that do not use indices, such as PROJECT operations, are not part of this model.) The input parameters, such as costs and user workload, are given per day. The output summary is computed per month. The rest is self-explanatory.

CONCLUDING REMARKS

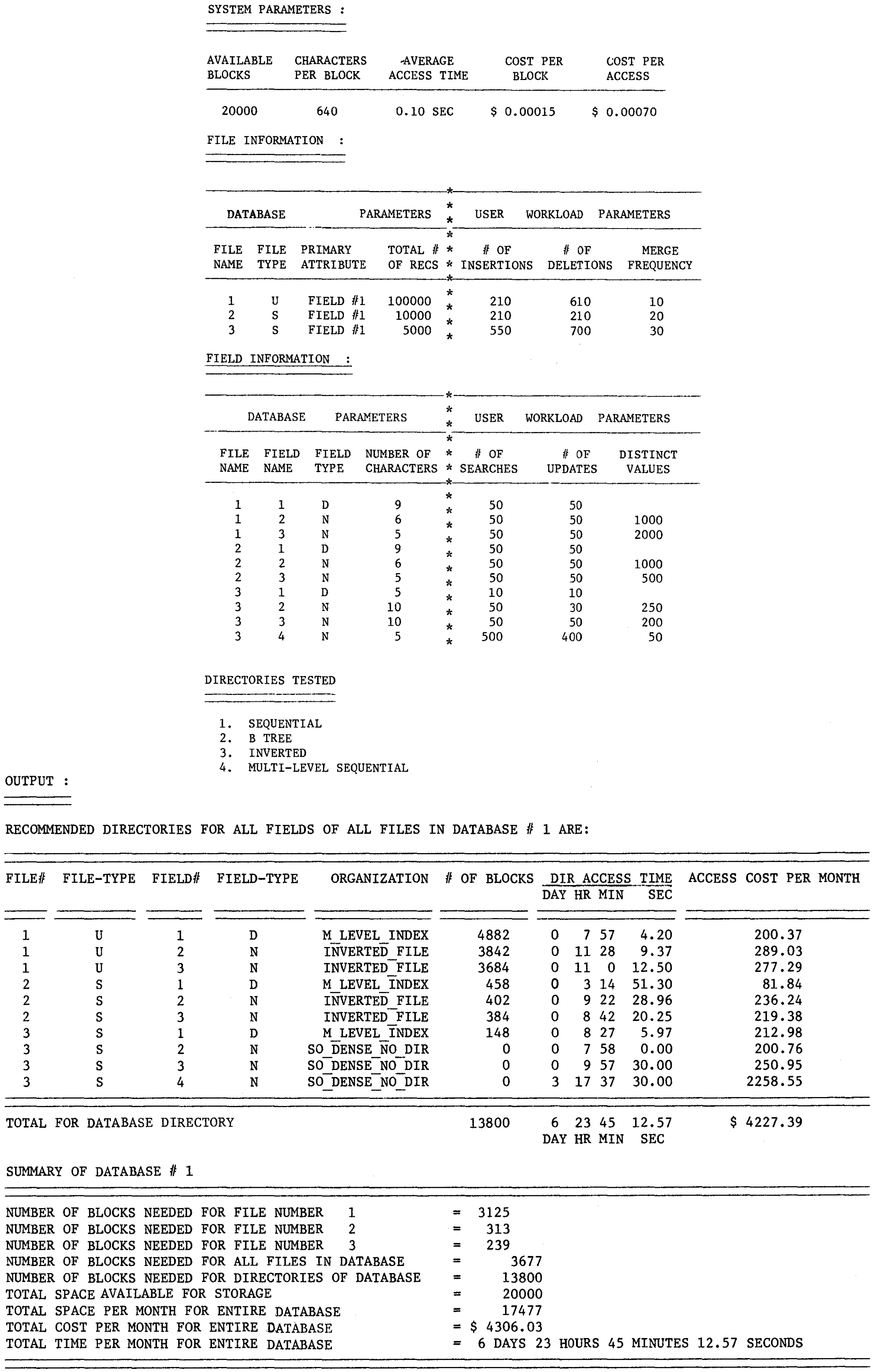
A model for prediction and optimization of the performance of relational databases has been developed. The model is concerned with selection of an optimal set of indices and reorganization points. It provides the total cost, time, and space associated with the selected indices for the given input parameters. It is a natural extension of previous work. It takes into consideration the effects of transaction on different fields and the total system's capacity and constraints, and it allows the user to evaluate indices that the user is interested in. The model distinguishes between primary field and secondary fields, between dense and nondense attributes, and between sorted and unsorted files. The model has been implemented by a PASCAL program, and sample simulation runs are provided. It is more complete than previous work, and it is easy to use. The complexity of the algorithm is 0(number of fields) · (number of evaluated indices +1)).

ACKNOWLEDGMENTS

The author wishes to thank Mr. Chung-Liang Lin for converting the algorithms into a PASCAL program and generating the tests. The author also wishes to thank Dr. Donna Kaminski for her valuable comments and suggestions.

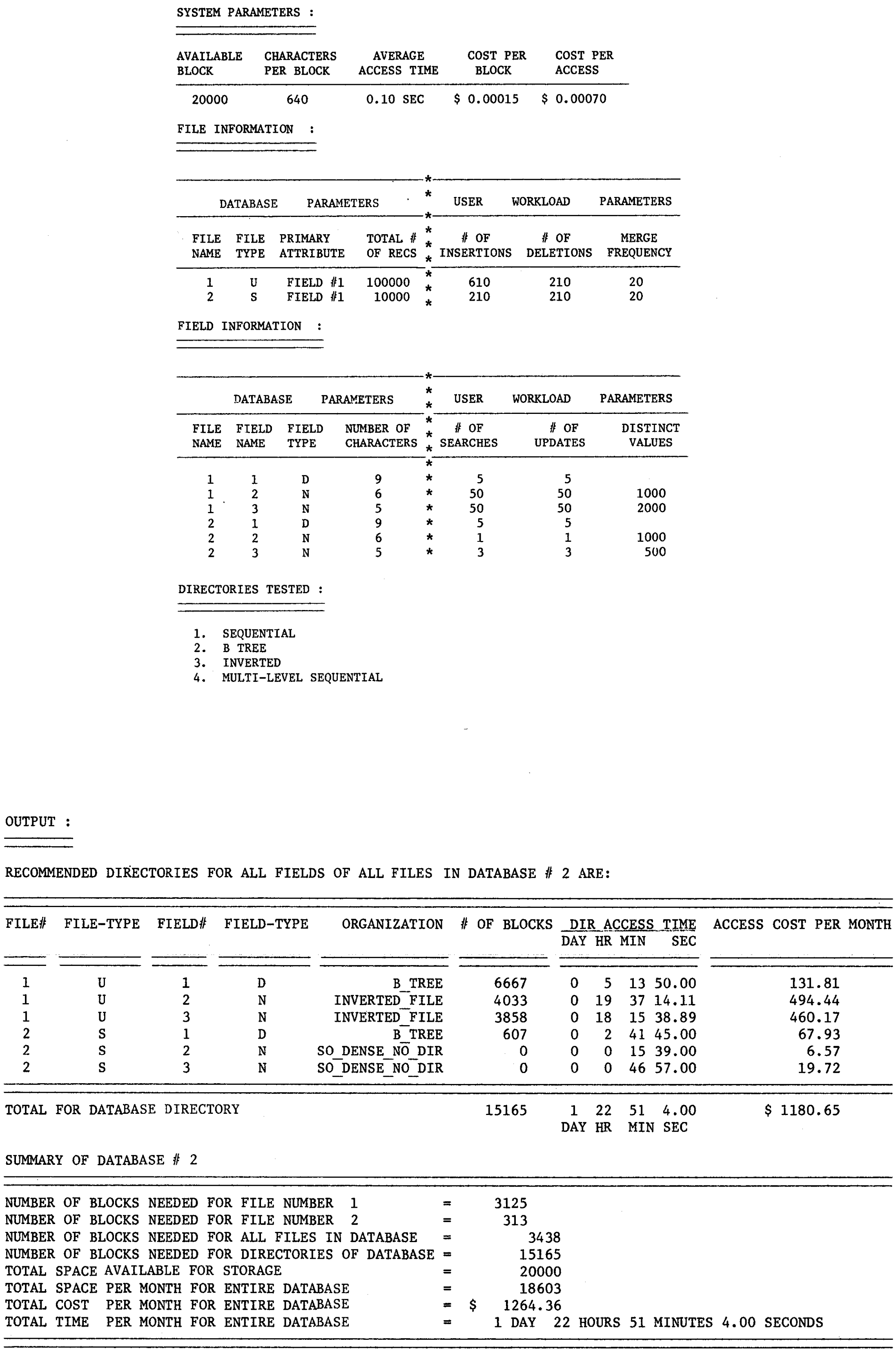
REFERENCES

1. Batory, D. S. "B$^+$ Trees and Indexed Sequential Files: A Performance Comparison." *ACM Proceedings of SIGMOD*. New York: ACM, 1981, pp. 30–39.
2. Batory, D. S. "Optimal File Designs and Reorganization Points." *ACM Transactions on Database Systems*, 7 (1982), pp. 60–81.
3. Batory, D. S., and C. C. Gotlieb. "A Unifying Model of Physical Databases." *ACM Transactions on Database Systems*, 7 (1982), pp. 509–538.
4. Bonfatti, F., D. Maio, and P. Tiberio. "A Separability-based Method for Secondary Index Selection in Physical Database Design." In *Methodology and Tools for Database Design*. Amsterdam: North-Holland, 1983,
5. Carlis, J. V., S. T. March, and G. W. Dickson. "Physical Database Design, a DSS Approach." *Information and Management*, 6 (1983), pp. 211–224.
6. Chen, P. P., and S. B. Yao. "Design and Performance Tools for Database Systems." *IEEE Proceedings of the International Conference on Very Large Data Bases*. New York: IEEE, 1977, pp. 3–15.
7. Christodoulakis, S. "Estimating Record Selectivities." *Information Systems*, 8 (1983), pp. 105–115.
8. Hoffer, J. A. "An Integer Programming Formulation of Computer Database Design Problems." *Information Science*, 11 (1976), pp. 29–48.
9. Horowitz, E., and S. Sahni. *Fundamentals of Data Structures*. Rockville, Md.: Computer Science Press, 1976.
10. Lum, V. Y., and H. Ling. "An Optimization Problem on the Selection of Secondary Keys." *Proceedings of ACM Annual Conference*. New York: ACM, 1971, pp. 349–356.
11. March, S. T., and D. G. Severance. "The Determination of Efficient Record Segmentation and Blocking Factors for Shared Data Files." *ACM Transactions on Database Systems*, 2 (1977) 3, pp. 279–296.
12. Mendelson, H. "Analysis of Extendible Hashing." *IEEE Transactions on Software Engineering*, SE-8 (1982) 6, pp. 611–619.
13. Motzkin, D., K. Williams, and K. Chang. "Uniform Organization of Inverted Files." *AFIPS, Proceedings of 1984 National Computer Conference* (Vol. 53), 1984, pp. 567–585.
14. Motzkin, D. "The Use of Normal Multiplication Tables For Information Storage and Retrieval." *Communication of the Association for Computing Machinery* (CACM), Vol. 22, (1979) 3, pp. 193–207.
15. Motzkin, D. "Computer Assisted Optimization and Prediction of Database Performance." Western Michigan University, Computer Science Department, Report 84-01, September 1984.
16. Nicolas, G. S. "A Generalized Database Access Path Model." *AFIPS Proceedings of the National Computer Conference*, 1981, pp. 529–535.
17. Schkolnick, M. "The Optimal Selection of Secondary Indices for Files." *Information Systems*, 1 (1975), pp. 141–146.
18. Schkolnick, M. "A Survey of Physical Database Design Methodology and Techniques." *Proceedings of the Fourth International Conference on Very Large Databases*. New York: IEEE, 1978, pp. 474–487.
19. Whang, K. W., G. Wiederhold, and D. Segalowics. "Separability—An Approach to Physical Database Design." *Proceedings of the Seventh International Conference on Very Large Databases*. New York: IEEE, 1982, pp. 320–332.
20. Yao, S. B., and A. G. Mertin. "Selection of File Organization Using an Analytic Model." *Proceedings of the International Conference on Very Large Databases*. New York: IEEE, 1975, pp. 255–267.
21. Yao, S. B., K. S. Das, and T. J. Theorey. "A Dynamic Database Reorganization Algorithm." *ACM Transactions on Database Systems*, 1, pp. 150–174.
22. Yao, S. B. "Modelling and Performance Evaluation of Physical Database Structures." *ACM Proceedings of ACM National Conference*. New York: ACM, 1976, pp. 303–309.
23. Yao, S. B. "An Attribute Based Model for Database Access Cost Analysis." *ACM Transactions on Database Systems*, 2 (1977), pp. 45–67.



```
                    SYSTEM PARAMETERS :

                    AVAILABLE   CHARACTERS    AVERAGE       COST PER      COST PER
                    BLOCKS      PER BLOCK    ACCESS TIME     BLOCK         ACCESS

                      20000        640        0.10 SEC     $ 0.00015     $ 0.00070

                    FILE INFORMATION  :

                    ----------------------------*----------------------------------
                                                *
                      DATABASE        PARAMETERS  *  USER    WORKLOAD   PARAMETERS
                    ----------------------------*----------------------------------
                                                *
                     FILE  FILE  PRIMARY    TOTAL # *   # OF        # OF       MERGE
                     NAME  TYPE  ATTRIBUTE  OF RECS * INSERTIONS  DELETIONS  FREQUENCY
                    ----------------------------*----------------------------------
                                                *
                      1     U     FIELD #1   100000 *     210         610         10
                      2     S     FIELD #1    10000 *     210         210         20
                      3     S     FIELD #1     5000 *     550         700         30

                    FIELD INFORMATION  :

                    ----------------------------*----------------------------------
                                                *
                        DATABASE     PARAMETERS   *  USER    WORKLOAD   PARAMETERS
                    ----------------------------*----------------------------------
                                                *
                      FILE  FIELD  FIELD  NUMBER OF  *  # OF        # OF      DISTINCT
                      NAME  NAME   TYPE   CHARACTERS * SEARCHES    UPDATES     VALUES
                    ----------------------------*----------------------------------
                                                *
                       1     1      D         9      *    50          50
                       1     2      N         6      *    50          50          1000
                       1     3      N         5      *    50          50          2000
                       2     1      D         9      *    50          50
                       2     2      N         6      *    50          50          1000
                       2     3      N         5      *    50          50           500
                       3     1      D         5      *    10          10
                       3     2      N        10      *    50          30           250
                       3     3      N        10      *    50          50           200
                       3     4      N         5      *   500         400            50

                    DIRECTORIES TESTED

                       1.  SEQUENTIAL
                       2.  B TREE
                       3.  INVERTED
                       4.  MULTI-LEVEL SEQUENTIAL
OUTPUT :

RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 1 ARE:

FILE#  FILE-TYPE  FIELD#  FIELD-TYPE    ORGANIZATION   # OF BLOCKS   DIR ACCESS TIME   ACCESS COST PER MONTH
                                                                   DAY HR MIN   SEC

  1        U        1         D         M_LEVEL_INDEX      4882      0   7 57    4.20          200.37
  1        U        2         N          INVERTED_FILE     3842      0  11 28    9.37          289.03
  1        U        3         N          INVERTED_FILE     3684      0  11  0   12.50          277.29
  2        S        1         D         M_LEVEL_INDEX       458      0   3 14   51.30           81.84
  2        S        2         N          INVERTED_FILE      402      0   9 22   28.96          236.24
  2        S        3         N          INVERTED_FILE      384      0   8 42   20.25          219.38
  3        S        1         D         M_LEVEL_INDEX       148      0   8 27    5.97          212.98
  3        S        2         N        SO_DENSE_NO_DIR        0      0   7 58    0.00          200.76
  3        S        3         N        SO_DENSE_NO_DIR        0      0   9 57   30.00          250.95
  3        S        4         N        SO_DENSE_NO_DIR        0      3  17 37   30.00         2258.55

TOTAL FOR DATABASE DIRECTORY                                  13800      6  23 45   12.57       $ 4227.39
                                                                       DAY HR MIN   SEC

SUMMARY OF DATABASE # 1

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER   1          =  3125
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER   2          =   313
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER   3          =   239
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE    =      3677
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE  =     13800
TOTAL SPACE AVAILABLE FOR STORAGE                    =     20000
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE            =     17477
TOTAL COST PER MONTH FOR ENTIRE DATABASE             = $ 4306.03
TOTAL TIME PER MONTH FOR ENTIRE DATABASE             =  6 DAYS 23 HOURS 45 MINUTES 12.57 SECONDS
```

Figure 2—Simulation run: Database 1



```
                    SYSTEM PARAMETERS :
                    ===================

                    AVAILABLE   CHARACTERS    AVERAGE      COST PER    COST PER
                    BLOCK       PER BLOCK    ACCESS TIME    BLOCK      ACCESS
                    -----------------------------------------------------------

                      20000         640        0.10 SEC   $ 0.00015   $ 0.00070

                    FILE INFORMATION  :
                    ===================

                    ------------------------------*-----------------------------
                                                  *
                         DATABASE     PARAMETERS   *    USER    WORKLOAD    PARAMETERS
                    ------------------------------*-----------------------------
                                                  *
                      FILE  FILE  PRIMARY    TOTAL #  *    # OF       # OF       MERGE
                      NAME  TYPE  ATTRIBUTE  OF RECS  * INSERTIONS  DELETIONS  FREQUENCY
                    ------------------------------------------------------------
                                                  *
                        1    U     FIELD #1   100000  *    610        210         20
                        2    S     FIELD #1    10000  *    210        210         20
                                                  *
                    FIELD INFORMATION  :
                    ====================

                    ------------------------------*-----------------------------
                                                  *
                           DATABASE     PARAMETERS  *    USER     WORKLOAD     PARAMETERS
                    ------------------------------*-----------------------------
                                                  *
                      FILE  FIELD  FIELD   NUMBER OF  *   # OF        # OF       DISTINCT
                      NAME  NAME   TYPE   CHARACTERS  * SEARCHES     UPDATES      VALUES
                    ------------------------------------------------------------
                                                  *
                        1     1     D          9      *    5            5
                        1     2     N          6      *   50           50          1000
                        1     3     N          5      *   50           50          2000
                        2     1     D          9      *    5            5
                        2     2     N          6      *    1            1          1000
                        2     3     N          5      *    3            3           500

                    DIRECTORIES TESTED :
                    ====================

                      1.  SEQUENTIAL
                      2.  B TREE
                      3.  INVERTED
                      4.  MULTI-LEVEL SEQUENTIAL


OUTPUT :
========

RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 2 ARE:

==========================================================================================

FILE#  FILE-TYPE  FIELD#  FIELD-TYPE    ORGANIZATION  # OF BLOCKS  DIR ACCESS TIME  ACCESS COST PER MONTH
                                                                 DAY HR MIN   SEC
=====  =========  ======  ==========  ==============  ===========  ===============  =====================

  1        U        1         D               B_TREE      6667      0   5  13 50.00          131.81
  1        U        2         N        INVERTED_FILE      4033      0  19  37 14.11          494.44
  1        U        3         N        INVERTED_FILE      3858      0  18  15 38.89          460.17
  2        S        1         D               B_TREE       607      0   2  41 45.00           67.93
  2        S        2         N       SO_DENSE_NO_DIR        0      0   0  15 39.00            6.57
  2        S        3         N       SO_DENSE_NO_DIR        0      0   0  46 57.00           19.72
==========================================================================================

TOTAL FOR DATABASE DIRECTORY                                15165      1  22  51  4.00       $ 1180.65
                                                                     DAY HR  MIN SEC

SUMMARY OF DATABASE # 2
==========================================================================================

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  1          =    3125
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  2          =     313
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE   =        3438
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE =       15165
TOTAL SPACE AVAILABLE FOR STORAGE                   =       20000
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE           =       18603
TOTAL COST  PER MONTH FOR ENTIRE DATABASE           =  $   1264.36
TOTAL TIME  PER MONTH FOR ENTIRE DATABASE           =    1 DAY  22 HOURS 51 MINUTES 4.00 SECONDS
==========================================================================================
```

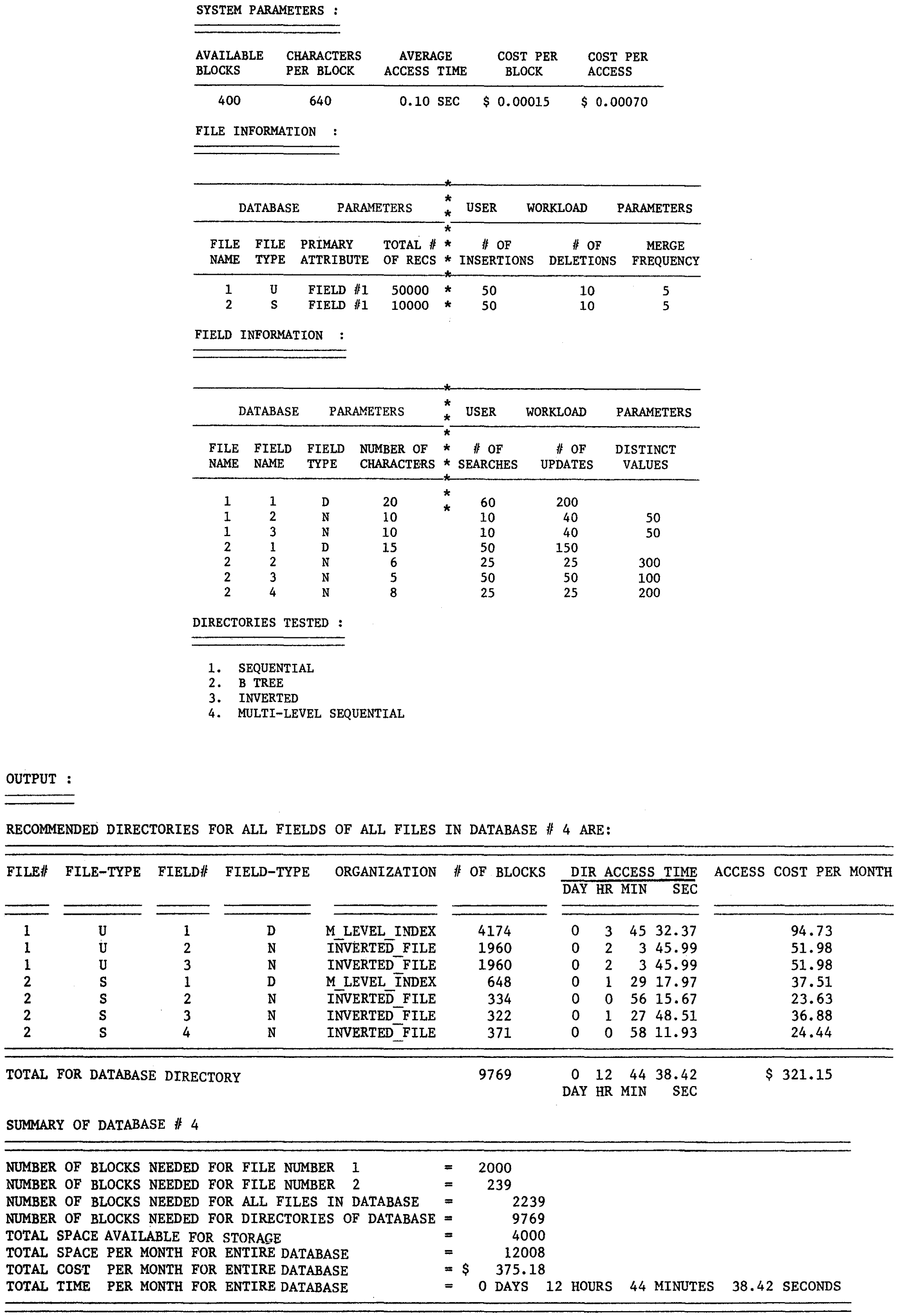
Figure 3—Simulation run: Database 2

```
          SYSTEM PARAMETERS :
          ===================

          AVAILABLE  CHARACTERS    AVERAGE     COST PER    COST PER
          BLOCK      PER BLOCK   ACCESS TIME    BLOCK      ACCESS
          ---------------------------------------------------------
           900         640        0.10 SEC    S 0.00015   $ 0.00070

          FILE INFORMATION  :
          ==================

          ------------------------------------*----------------------------------
                                              *
                DATABASE    PARAMETERS        *  USER    WORKLOAD    PARAMETERS
          ------------------------------------*----------------------------------
            FILE  FILE  PRIMARY    TOTAL #    *   # OF       # OF        MERGE
            NAME  TYPE  ATTRIBUTE  OF RECS    * INSERTIONS DELETIONS  FREQUENCY
          ------------------------------------*----------------------------------
             1     U     FIELD #1   1000      *    60         60          20
             2     S     FIELD #1   9000      *   100         75          10

          FIELD INFORMATION  :
          ===================

          ------------------------------------*----------------------------------
                                              *
              DATABASE      PARAMETERS        *  USER    WORKLOAD    PARAMETERS
          ------------------------------------*----------------------------------
          FILE  FIELD  FIELD    NUMBER OF     *   # OF       # OF    DISTINCT
          NAME  NAME   TYPE     CHARACTERS    * SEARCHES   UPDATES    VALUES
          ------------------------------------*----------------------------------
                                              *
            1     1     D          6          *   60         60
            1     2     D          6          *   10         10
            1     3     N          6          *   10         10          50
            2     1     D          9          *   50         50
            2     2     D          9          *   25         25
            2     3     N          3          *   50         50         500
            2     4     N          4          *   25         25         600
                                              *

          DIRECTORIES TESTED :
          ====================

            1.  SEQUENTIAL
            2.  B TREE
            3.  INVERTED
            4.  MULTI-LEVEL SEQUENTIAL


OUTPUT :
========

RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 3 ARE:
==========================================================================================

FILE #  FILE-TYPE  FIELD#  FIELD-TYPE  ORGANIZATION    # OF BLOCKS  DIR ACCESS TIME   ACCESS COST PER MONTH
                                                                    DAY HR MIN  SEC
======  =========  ======  ==========  ==============  ===========  ===============   =====================

  1         U        1         D       M_LEVEL_INDEX       34        0  1   3 26.10                26.64
  1         U        2         D       M_LEVEL_INDEX       34        0  0  45 56.10                19.29
  1         U        3         N       RANDOM_NO_DIR        0        0  0   7 45.00                 3.25
  2         S        1         D       M_LEVEL_INDEX      378        0  1  40  4.80                42.03
  2         S        2         D       M_LEVEL_INDEX      378        0  1  28 49.80                37.31
  2         S        3         N       INVERTED_FILE      224        0  2  17 39.06                57.81
  2         S        4         N       INVERTED_FILE      240        0  1  42  7.87                42.90
==========================================================================================================

TOTAL FOR DATABASE DIRECTORY                              1288       0  9   5 48.73              $ 229.24
                                                                    DAY HR MIN  SEC

SUMMARY OF DATABASE #3
==========================================================================================================

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  1          =        29
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER 2           =       360
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE   =          389
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE =         1288
TOTAL SPACE AVAILABLE FOR STORAGE                   =          900
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE           =         1677  ***
TOTAL COST  PER MONTH FOR ENTIRE DATABASE           =    $  236.79
TOTAL TIME  PER MONTH FOR ENTIRE DATABASE           =      0 DAYS  9 HOURS  5 MINUTES  48.73 SECONDS
==========================================================================================================
```

Figure 4—Simulation run: Database 3

```
DATABASE EXCEEDS AVAILABLE SPACE, THE FOLLOWING ADJUSTMENTS HAVE BEEN MADE :
==========================================================================

   FILE #    DELETED DIR FIELD #    SPACE SAVED     SPACE NEEDED FOR NEW DATABASE
   ======    ===================    ===========     =============================

     2                2             378-                     1299
     2                1             378-                      921
     1                2              34-                      887
   ......    ...................    ...........     .............................

AFTER ADJUSTMENT, THE RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 3 ARE:
=============================================================================================

FILE#  FILE-TYPE  FIELD#  FIELD-TYPE    ORGANIZATION  # OF BLOCKS  DIR ACCESS TIME  ACCESS COST PER MONTH
                                                                   DAY HR MIN   SEC
=====  =========  ======  ==========  ============  ===========  ===============  =====================

  1        U         1         D       M_LEVEL_INDEX        34      0  1   3 26.10                26.64
  1        U         2         D       RANDOM_NO_DIR         0      0  2  10 15.00                54.70
  1        U         3         N       RANDOM_NO_DIR         0      0  0   7 45.00                 3.25
  2        S         1         D     SO_DENSE_NO_DIR         0      0  2  21 18.00                59.35
  2        S         2         D     SO_DENSE_NO_DIR         0      0  1  58 48.00                49.90
  2        S         3         N       INVERTED_FILE       224      0  2  17 39.06                57.81
  2        S         4         N       INVERTED_FILE       240      0  1  42  7.87                42.90
=============================================================================================

TOTAL FOR DATABASE DIRECTORY                               498      0 11  41 19.03             $ 294.55

SUMMARY OF DATABASE # 3
===================================================================================

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  1          =      29
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  2          =     360
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE   =         389
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE =         498
TOTAL SPACE AVAILABLE FOR STORAGE                   =         900
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE           =         887
TOTAL COST  PER MONTH FOR ENTIRE DATABASE           = $    298.54
TOTAL TIME  PER MONTH FOR ENTIRE DATABASE           =   0 DAYS  11 HOURS  41 MINUTES  19.03 SECONDS
===================================================================================
```

Figure 4—(continued)



```
                    SYSTEM PARAMETERS :
                    ===================

                    AVAILABLE   CHARACTERS    AVERAGE      COST PER     COST PER
                    BLOCKS      PER BLOCK    ACCESS TIME    BLOCK       ACCESS
                    ------------------------------------------------------------
                      400          640        0.10 SEC    $ 0.00015    $ 0.00070
                    FILE INFORMATION  :
                    ===================

                    -------------------------------*------------------------------
                          DATABASE      PARAMETERS   *  USER     WORKLOAD    PARAMETERS
                    -------------------------------*------------------------------
                      FILE  FILE  PRIMARY    TOTAL # *   # OF        # OF       MERGE
                      NAME  TYPE  ATTRIBUTE  OF RECS * INSERTIONS  DELETIONS  FREQUENCY
                    -------------------------------*------------------------------
                       1     U     FIELD #1   50000  *   50          10          5
                       2     S     FIELD #1   10000  *   50          10          5
                    FIELD INFORMATION  :
                    ====================

                    -------------------------------*------------------------------
                          DATABASE     PARAMETERS    *  USER     WORKLOAD    PARAMETERS
                    -------------------------------*------------------------------
                      FILE  FIELD  FIELD  NUMBER OF  *   # OF        # OF      DISTINCT
                      NAME  NAME   TYPE   CHARACTERS * SEARCHES    UPDATES     VALUES
                    -------------------------------*------------------------------
                       1     1      D        20      *   60         200
                       1     2      N        10      *   10          40          50
                       1     3      N        10          10          40          50
                       2     1      D        15          50         150
                       2     2      N         6          25          25         300
                       2     3      N         5          50          50         100
                       2     4      N         8          25          25         200
                    DIRECTORIES TESTED :
                    ====================

                       1.  SEQUENTIAL
                       2.  B TREE
                       3.  INVERTED
                       4.  MULTI-LEVEL SEQUENTIAL


OUTPUT :
========

RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 4 ARE:
==========================================================================

FILE#  FILE-TYPE  FIELD#  FIELD-TYPE   ORGANIZATION  # OF BLOCKS   DIR ACCESS TIME  ACCESS COST PER MONTH
                                                                   DAY HR MIN  SEC
=====  =========  ======  ==========  ============  ===========   ===============  =====================

  1        U        1         D       M_LEVEL_INDEX     4174        0  3  45 32.37           94.73
  1        U        2         N       INVERTED_FILE     1960        0  2   3 45.99           51.98
  1        U        3         N       INVERTED_FILE     1960        0  2   3 45.99           51.98
  2        S        1         D       M_LEVEL_INDEX      648        0  1  29 17.97           37.51
  2        S        2         N       INVERTED_FILE      334        0  0  56 15.67           23.63
  2        S        3         N       INVERTED_FILE      322        0  1  27 48.51           36.88
  2        S        4         N       INVERTED_FILE      371        0  0  58 11.93           24.44
==========================================================================================================

TOTAL FOR DATABASE DIRECTORY                            9769        0 12  44 38.42        $ 321.15
                                                                   DAY HR MIN  SEC

SUMMARY OF DATABASE # 4
=====================================================================================

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  1           =   2000
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  2           =    239
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE    =       2239
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE  =       9769
TOTAL SPACE AVAILABLE FOR STORAGE                    =       4000
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE            =      12008
TOTAL COST  PER MONTH FOR ENTIRE DATABASE            = $   375.18
TOTAL TIME  PER MONTH FOR ENTIRE DATABASE            =   0 DAYS  12 HOURS  44 MINUTES  38.42 SECONDS
=====================================================================================
```

Figure 5—Simulation run: Database 4

```
DATABASE EXCEEDS AVAILABLE SPACE, THE FOLLOWING ADJUSTMENTS HAVE BEEN MADE :
____________________________________________________________________________

   FILE #     DELETED DIR FIELD #     SPACE SAVED     SPACE NEEDED FOR NEW DATABASE
   ______     ___________________     ___________     _____________________________

     2                1                  648-                  11360
     2                4                  371-                  10989
     2                2                  334-                  10655
     1                3                 1960-                   8695
     1                2                 1960-                   6735
     2                3                  322-                   6413
     1                1                 4174-                   2239
     2                3                  322+                   2561
     2                2                  334+                   2895
     2                4                  371+                   3266
     2                1                  648+                   3914
 .....  ...................     ...........     .............................

AFTER ADJUSTMENT, THE RECOMMENDED DIRECTORIES FOR ALL FIELDS OF ALL FILES IN DATABASE # 4 ARE:
_____________________________________________________________________________________________

FILE#  FILE-TYPE  FIELD#  FIELD-TYPE    ORGANIZATION  # OF BLOCKS   DIR ACCESS TIME  ACCESS COST PER MONTH
                                                                    DAY HR MIN  SEC
_____  _________  ______  __________  ____________  ___________  ________________  _____________________

  1        U        1         D        RANDOM_NO_DIR         0      12   4   1 30.00            7359.03
  1        U        2         N        RANDOM_NO_DIR         0       0   8  22  0.00             210.84
  1        U        3         N        RANDOM_NO_DIR         0       0   8  22  0.00             210.84
  2        S        1         D        M_LEVEL_INDEX       648       0   1  29 17.97              37.51
  2        S        2         N        INVERTED_FILE       334       0   0  56 15.67              23.63
  2        S        3         N        INVERTED_FILE       322       0   1  27 48.51              36.88
  2        S        4         N        INVERTED_FILE       371       0   0  58 11.93              24.44
_____________________________________________________________________________________________________

TOTAL FOR DATABASE DIRECTORY                                  1675      13   1  37  4.06         $ 7903.17
                                                                        DAY HR MIN  SEC
SUMMARY OF DATABASE # 4
_____________________________________________________________________________________________________

NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  1         =     2000
NUMBER OF BLOCKS NEEDED FOR FILE NUMBER  2         =      239
NUMBER OF BLOCKS NEEDED FOR ALL FILES IN DATABASE  =         2239
NUMBER OF BLOCKS NEEDED FOR DIRECTORIES OF DATABASE =        1675
TOTAL SPACE AVAILABLE FOR STORAGE                  =         4000
TOTAL SPACE PER MONTH FOR ENTIRE DATABASE          =         3914
TOTAL COST  PER MONTH FOR ENTIRE DATABASE          =  $   7920.78
TOTAL TIME  PER MONTH FOR ENTIRE DATABASE          =    13 DAYS  1 HOURS  37 MINUTES  4.06 SECONDS
_____________________________________________________________________________________________________
```

Figure 5—(continued)

A general concurrency control for database systems

by ABDEL A. FARRAG and M. TAMER OZSU
The University of Alberta
Edmonton, Alberta

ABSTRACT

The *concurrency control* problem in database systems has been examined by many people and several concurrency control algorithms have been proposed. The most popular concurrency controls are *two-phase locking* and *timestamp ordering*. This paper reviews both algorithms and shows that they are special cases of a more general concurrency control algorithm. This concurrency control algorithm is described in detail and is proven to work correctly. Several other special cases of the proposed mechanism are presented. We show that two-phase locking and timestamp ordering represent the two end points of a series of concurrency controls. Each of them is a special case of the general concurrency control described in this paper.

INTRODUCTION

The problem of coordinating concurrent accesses to a database system has been studied by many people and several concurrency control algorithms have been introduced.[1–5] The task of a concurrency control is to ensure the consistency of the database while allowing a set of transactions to execute concurrently. A database is a collection of entities named *x*, *y*, *z*, etc. Each entity has a single value. The consistency of the database is defined in terms of a set of constraints called integrity constraints. The database is said to be consistent if the values of its entities satisfy the predefined set of integrity constraints. A transaction is any sequence of atomic actions which preserves the consistency of the database. Each action is either a read or write operation on some entity of the database. In order to allow a set of transactions to execute concurrently, a concurrency control is needed to resolve the conflicts among transactions and to ensure that the overall effect of their execution is correct (i.e., transforms a consistent database state into a new consistent state).

Serializability still remains the main correctness criterion for concurrency control. A concurrent execution of a set of transactions (or *schedule*) is said to be serializable if it is equivalent to (i.e., has the same effect on the database as) a serial schedule in which the transactions are exeucted sequentially in some order. The concurrency control performs its job by producing only serializable schedules.

The most popular (and practical) concurrency controls are *two-phase locking*[1,6,7] and *timestamp ordering*.[3,4] This paper reviews both mechanisms and shows that they are special cases of a more general concurrency control algorithm. This concurrency control algorithm is described in detail and is proven to work correctly. Several other special cases of the proposed mechanism are also presented. The set of transactions executing at any time under the proposed concurrency control forms a set of classes called *strict classes*. The maximum number of transactions which can belong to the same strict class is called the *strictness level* of the concurrency control. The value of the strictness level can be specified in advance and can be modified during execution. The higher the value of the strictness level the more strict the concurrency control and the lower the value of the strictness level the less strict the concurrency control.

TWO-PHASE LOCKING AND TIMESTAMP ORDERING

This section defines some basic concepts which will be used throughout this paper and reviews two-phase locking and timestamp ordering mechanisms.

Definition 1 (Transaction): A transaction is any sequence of read and/or write operations which preserves the integrity constraints of the database. A read operation by a transaction T_i on an entity x is denoted $R_i(x)$. Similarly, a write operation on x by T_i is denoted $W_i(x)$. A transaction is assumed to be a correct computation (i.e., either *all* of its updates must be reflected in the database or *none* of them). A transaction is said to be terminated or committed when all of its operations have been accepted by the concurrency control. If an operation by the transaction is rejected, the transaction must be aborted.

Definition 2 (Schedule): A schedule S of a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is an interleaved sequence of the operations of the transactions in T. When the operations of each transaction appear consecutively (i.e., without interleaving with the operations of the other transactions), the schedule is said to be *serial.*

Definition 3 (Conflicting operations): Two operations by two different transactions are said to be in conflict iff both operations access the same entity and one of them is a write operation. This definition implies that read operations do not conflict.

Definition 4 (Dependency graph): The dependency graph DG(S) of a schedule S of a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is a directed graph which represents the conflict relations among transactions. The set of nodes of this graph is $\{T_1, T_2, \ldots, T_n\}$, and an arc form T_i to T_j exists in the graph iff there is an operation by T_i which conflicts with and precedes (i.e., its order in the schedule comes before) another operation by T_j.

Definition 5 (Equivalence of two schedules): Two schedules S_1 and S_2 for the same set of transactions are said to be equivalent if $DG(S_1) = DG(S_2)$. A schedule S is said to be serializable iff it is equivalent to a serial schedule.

Theorem 1

If the dependency graph DG(S) of a schedule S is acyclic, then S is serializable.[1,8]

Two-Phase Locking

Locking is the most commonly used mechanism for controlling concurrency. Locking requires each transaction before reading or writing an entity *x* to obtain a read-lock or a write-lock on *x*, respectively. Several transactions can obtain a read-lock on *x* simultaneously, but no more than one transaction can obtain a write-lock on *x* at the same time. When a read- or write-lock on some entity cannot be granted because the entity is already locked, the transaction which requested the lock has to wait until the lock is released.

The most famous locking protocol is called *two-phase locking.*[1] Two-phase locking guarantees serializability by preventing a transaction from obtaining a lock on any entity after releasing a lock on any other entity. Therefore, each transaction has two phases, one during which locks can only be obtained followed by another one during which locks can only be released. The point at which the transaction releases its first lock delimits the two phases. This point is called the *locked point* of the transaction.

Some systems do not require a transaction to use explicit commands (like Lock(x) and Unlock(x)) to lock and release the entities it accesses. Instead, a read-lock or a write-lock is requested implicitly when the transaction submits a read or write operation, respectively, to the concurrency control. This lock will be granted if the operation is accepted; otherwise the operation is placed on a waiting queue. When all the operations of the transaction succeed (i.e., have been accepted), all locks are released by a single atomic action. This form of two-phase locking is called *strict* two-phase locking[5,6] because each transaction maintains all locks until termination. Throughout this paper, the term two-phase locking will refer to a strict two-phase locking mechanism in which read- and write-locks are granted and released as described above.

Two-phase locking has some drawbacks. Because transactions can be arbitrarily long (but finite) and each transaction must maintain all the locks it obtains during execution until its very end, two-phase locking reduces the level of concurrency. Another drawback is the problem of *deadlock.*[7,9] Deadlock can occur because transactions wait for one another. In order to detect deadlock, a directed graph called the *wait for graph* is maintained. The nodes of the graph represent the transactions and the arcs represent the wait-for relationship (i.e., an arc $T_i \rightarrow T_j$ exists if T_i is waiting for T_j to release its locks on some entity. A deadlock occurs when the graph contains a cycle. In order to resolve the problem one or more transaction must be aborted.

<u>Theorem 2</u>

Two-phase locking is a correct synchronization mechanism (i.e., every schedule produced by a two-phase locking mechanism is serializable[1,8]).

Timestamp Ordering

In a timestamp ordering mechanism, each transaction is assigned a unique timestamp (i.e., number) when it starts. The timestamp of T_i will be attached to every read or write operation issued by T_i. For each entity x, two values are recorded, TSR(x) and TSW(x). These values record the largest timestamp of any read and write operations processed on x, respectively. Timestamp ordering guarantees serializability for executing all conflicting operations in timestamp order.

The basic timestamp ordering mechanism processes read and write operations as follows. When a read request $R_i(x)$ by a transaction T_i is received, the timestamp of T_i is compared with the value of TSW(x). If it is smaller, the operation is rejected. Otherwise, the operation is accepted. Similarly, when a write operation $W_i(x)$ is received, its timestamp is compared with the value max(TSR(x), TSW(x)). If it is smaller, the operation is rejected. Otherwise, it is accepted.

When an operation is rejected, the transaction which issued the operation must be aborted. Aborting a transaction involves undoing all steps which have been executed and restarting the transaction from the beginning. Abortion is the main drawback of the basic timestamp ordering. If it occurs frequently, performance can be degraded. Other timestamp ordering mechanisms which necessitate lesser abortion than the basic timestamp ordering mechanism described above have been proposed,[4,10,11] but in this paper, we are only concerned with the basic timestamp ordering.

The basic timestamp ordering mechanism has the advantage of being deadlock free. It also achieves a higher level of concurrency than two-phase locking because transactions never "wait."

<u>Theorem 3</u>

Timestamp ordering is a correct synchronization mechanism.[12]

THE PROPOSED CONCURRENCY CONTROL

This section describes the proposed concurrency control and proves it works correctly (i.e., produces only serializable schedules).

Transaction Timestamps

Each transaction T_i is assigned a unique timestamp $t_s(T_i)$, when it starts. The timestamp $t_s(T_i)$ is a pair $(t_g(T_i), t_l(T_i))$, where $t_g(T_i)$ is called the *global timestamp* of T_i and $t_l(T_i)$ is called the *local timestamp* of T_i. Timestamps are assigned using the method described later. The method assumes that there is some positive integer L representing the maximum number of transactions which can have the same global timestamp simultaneously. We will call L the *strictness level* of the concurrency control for a technical reason which will become apparent later. For the purpose of this section, we assume that the value of L is fixed (i.e., does not change during execution). How the value of L is chosen and its impact on the performance of the concurrency control will be discussed in the next section. In this section, we assume an arbitrary value for L.

The method used for assigning timestamps uses four counters, C_1, C_2, C_3 and C_4. The current value of each counter C_i is denoted by $V(C_i)$. C_1 and C_2 are used to generate the global and local timestamps, respectively. C_3 is used to count the number of transactions currently executing and having the global timestamp equal to $V(C_1)$. C_4 is used to keep track of the current number of executing transactions. $V(C_4)$ must always be $\leq M$ where M is some integer representing the multiprogramming level of the system (i.e., the maximum number of transactions which can be executed concurrently). It is assumed that if the number of executing transactions equals M, then no other transaction can start execution until one of the currently executing transactions is terminated or aborted. Assigning timestamps proceeds as described below. (Initially, all counters contain the value 0).

Starting or restarting a transaction

When a new transaction T_i arrives in the system (or when T_i is restarted) the values of $t_g(T_i)$ and $t_l(T_i)$ are assigned as follows:

1. If $V(C_4) = M$, then T_i cannot be started (i.e., put T_i on a waiting queue and stop). Otherwise, $V(C_4) = V(C_4) + 1$.
2. If $V(C_3) < L$, then $V(C_3) = V(C_3) + 1$ and $V(C_2) = V(C_2) + 1$. Otherwise, $V(C_3) = V(C_2) = 1$ and $V(C_1) = V(C_1) + 1$.
3. $t_g(T_i) = V(C_1)$ and $t_l(T_i) = V(C_2)$.

Terminating or aborting a transaction

When a transaction T_i is terminated or aborted, the values of $V(C_3)$ and $V(C_4)$ will be modified as follows:

1. If $t_g(T_i) = V(C_1)$ at the time of termination or abortion, then $V(C_3) = V(C_3) - 1$.
2. $V(C_4) = V(C_4) - 1$.

For each entity x, the concurrency control maintains the following values:

i) GTSW(x) and LTSW(x)

GTSW(x) is a variable which records the largest global timestamp of any transaction that wrote x. This value is recorded when the write operation of the transaction is accepted. The local timestamp of such a transaction is recorded at the same time in the list LTSW(x). If the transaction is terminated (or aborted) and its global and local timestamps are still recorded in GTSW(x) and LTSW(x), respectively, the local timestamp is deleted from LTSW(x).

ii) GTSR(x) and LTSR(x)

GTSR(x) is a variable which records the largest global timestamp of any transaction that read x. This value is recorded when the read operation of the transaction is accepted. The local timestamp of the transaction is recorded at the same time in the list LTSR(x). In general, several transactions with the same global timestamp (or with different global timestamps) can read x simultaneously. LTSR(x) records the local timestamp of every transaction that read x and whose global timestamp is recorded in GTSR(x). If a transaction is terminated (or aborted) and its global and local timestamps are recorded in GTSR(x) and LTSR(x), respectively, the local timestamp is deleted from LTSR(x). (Note that LTSR(x) may contain several values, each representing the local timestamp of some transaction.) Initially, i.e., before starting execution, LTSW(x) and LTSR(x) are empty, and GTSW(x) and GTSR(x) record the initial value of the counter C_i.

The following subsections describe how the concurrency control processes read and write operations and prove that the concurrency control produces only serializable schedules.

Processing Read Operations

When the concurrency control receives a read operation $R_i(x)$, one of the following cases arises:

1. $t_g(T_i) < \text{GTSW}(x)$. This means that x has been written by a transaction which has a larger global timestamp than T_i. In this case, $R_i(x)$ is rejected.
2. $t_g(T_i) > \text{GTSW}(x)$ This means that x has not been written by any transaction which has a larger global timestamp than T_i. In this case, $R_i(x)$ is accepted.
3. $t_g(T_i) = \text{GTSW}(x)$. This means that x has been written by a transaction which has the same global timestamp as T_i. When this case occurs, one of the following conditions is true:
   a. LTSW(x) is empty. This means that any transaction that wrote x and has the same global timestamp as T_i has been terminated. In this case $R_i(x)$ is accepted.
   b. LTSW(x) is not empty. This means that x has been written by another transaction which has the same global timestamp as T_i and that this transaction has not been terminated. In this case $R_i(x)$ has to wait.

In general, when a read or write operation has to wait because it conflicts with a previously granted operation by a different transaction having the same global timestamp, the operation cannot be processed until the transaction is terminated or aborted (and, therefore, the conflict no longer exists). If an operation with a larger global timestamp than the waiting operation and in conflict with it has been accepted, the waiting operation will be rejected (i.e., deleted from the waiting queue). The wait-for relationship among transactions can be represented by a directed graph similar to the one described previously.

Processing Write Operations

When the concurrency control receives a write operation $W_i(x)$, one of the following cases arises:

1. $t_g(T_i) < \max(\text{GTSR}(x), \text{GTSW}(x))$. This means that x has been read or written by any other transaction which has a larger global timestamp than T_i. In this case, $W_i(x)$ is rejected.
2. $t_g(T_i) > \max(\text{GTSR}(x), \text{GTSW}(x))$. This means that x has not been read or written by any other transaction which has a larger global timestamp than T_i. In this case, $W_i(x)$ is accepted.
3. $t_g(T_i) = \max(\text{GTSR}(x), \text{GTSW}(x))$. This means that x has been read or written by one or more transactions

having the same global timestamp as T_i. When this case occurs, one of the following conditions is true:

a. $GTSW(x) > GTSR(x)$. In this case, the concurrency control examines $LTSW(x)$. If $LTSW(x)$ is empty, then $W_i(x)$ is accepted; otherwise, $W_i(x)$ has to wait.
b. $GTSW(x) < GTSR(x)$. In this case, the concurrency control examines $LTSR(x)$. If $LTSR(x)$ is empty or contains only the local timestamp of T_i, then $W_i(x)$ is accepted; otherwise, $W_i(x)$ has to wait.
c. $GTSW(x) = GTSR(x)$. In this case, the concurrency control examines $LTSW(x)$ and $LTSR(x)$. If $LTSW(x)$ is empty and $LTSR(x)$ is empty or contains only the local timestamp of T_i, then $W_i(x)$ is accepted; otherwise, $W_i(x)$ has to wait.

Theorem 4

Every schedule produced by the concurrency control is serializable.

Proof: Let S be a schedule produced by the concurrency control. An arc $T_k \rightarrow T_l$ in DG(S) indicates that one of the following two conditions is true: either $t_g(T_k) < t_g(T_l)$ or $t_g(T_k) = t_g(T_l)$ and T_k terminates before T_l. Let T_i and T_j be two arbitrary nodes such that there is a path (of length greater than zero) from T_i to T_j in DG(S). Then, by transitivity, either $t_g(T_i) < t_g(T_j)$ or $t_g(T_i) = t_g(T_j)$ and T_i terminates before T_j. In either case, another path from T_j to T_i cannot exist because it leads to a contradiction. Therefore, DG(S) is acyclic (i.e., S is serializable).□

THE STRICTNESS LEVEL

The set of transactions executing at any time under the concurrency control can be partitioned into a set of disjoint classes; each class containing the set of transactions that have the same global timestamp. Let us call these classes *strict classes*. The set of strict classes changes dynamically during execution (i.e., when a transaction is started, terminated or aborted). In the previous section, we assumed that the maximum number of transactions that can have the same global timestamp simultaneously during execution (i.e., can belong to the same strict class) is chosen arbitrarily. We called this number the *strictness level* of the concurrency control. This section describes the impact of the strictness level on the performance of the concurrency control.

Consider a special case of the concurrency control described in the previous section in which the value of the strictness level $L \geq M$, where M is the multiprogramming level of the system (defined in the previous section). In this case, the set of executing transactions will always have the same global timestamp (i.e., the number of strict classes at any time during execution will equal 1) because the number of executing transactions cannot exceed the value of M. It is not difficult to see that in this particular case processing read and write operations is performed by the concurrency control as in the two-phase locking mechanism described earlier. As in the procedures described in the previous section for processing read and write operations, the conditions for rejecting an operation will never be satisfied. In this case, the concurrency control will respond to each read or write operation by either accepting or delaying the operation depending on the condition satisfied when the operation is received.

Consider another special case in which $L = 1$. In this case, the set of executing transactions will always have different global timestamps (i.e., the number of strict classes at any time during execution will equal the number of transactions executing at that time). It is not difficult to see that in this particular case processing read and write operations is performed by the concurrency control as in the timestamp ordering mechanism described earlier. As in the procedures described previously for processing read and write operations, the conditions for delaying an operation will never be satisfied. In this case, the concurrency control will respond to each read or write operation by either accepting or rejecting the operation depending on the condition satisfied when the operation is received. This proves the following lemma.

Lemma 1

Two-phase locking and timestamp ordering are special cases of the concurrency control mechanism proposed in this paper.

Several other special cases of the proposed mechanism arise for $L = 2, 3, \ldots, M - 1$. Processing read and write operations in each of these cases is performed by the concurrency control as if it were a combination of both mechanisms, two-phase locking and timestamp ordering. In this case, an operation will be processed relative to another operation having the same global timestamp as if the concurrency control is two-phase locking, and relative to another operation having a different global timestamp as if the concurrency control is timestamp ordering. The following lemma gives the minimum and the maximum number of strict classes at any time during execution for any value of L, where $1 \leq L \leq M$.

Lemma 2

Let L, M, $V(C_4)$ and C refer to the strictness level, the multiprogramming level, the number of executing transactions and the number of strict classes at any given time during execution, respectively. Then, $[V(C_4)/L] \leq C \leq \min(V(C_4), M - L + 1)$.

Proof: The proof of the above lemma is an immediate consequence of the method described previously for assigning timestamps. The minimum number of strict classes corresponds to a situation in which no more than one class has fewer than L transactions. The maximum number of strict classes corresponds to a situation in which M-L transactions (or every transaction if $V(C_4) \leq M - L$) will belong to $M - L(V(C_4))$ different classes and the remaining transactions (if any) will belong to another strict class. (The reader must convince himself that the above situations can occur.) □

Therefore, two-phase locking and timestamp ordering represent the two end points of a series of concurrency controls.

Each of them is a special case of the general concurrency control mechanism described in this paper (i.e., for every different value of L, where $1 \leq L \leq M$, a different special case results). Each of these special cases has a different level of concurrency. In particular, two-phase locking and timestamp ordering have the lowest and highest level of concurrency, respectively. The value of the strictness level is, in some sense, a measure of the level of concurrency. The greater the value of L, the stricter the concurrency control. Although we assumed in the previous section that the value of the strictness level is fixed during execution, this is not necessary in fact. The value *can* change from time to time during execution (without affecting the correctness of the concurrency control).

Some researchers[2] have already shown that if the level of transaction conflict is low, other concurrency control mechanisms which allow a higher level of concurrency will perform better than locking. We believe that the performance of a concurrency control mechanism must depend on both the likelihood of transaction conflict and the level of concurrency allowed by the mechanism. The exact relationship between performance and transaction conflict and concurrency has not been formalized and deserves more research.

This paper has proposed a general concurrency control mechanism in which the strictness level (or, in some sense, the concurrency level) can be specified and even modified during execution. The performance issue, which has not been examined in detail, is the subject of further research. A problem which has not been discussed is that of *deadlock*. Deadlock can only occur if the strictness level is greater than 1. Moreover, it only involves transactions which have the same global timestamp because a transaction never waits for any other transaction which has differnt global timestamp. Deadlock can be detected by maintaining a wait-for graph similar to the one described previously.

CONCLUSIONS

Two-phase locking and timestamp ordering are the most popular and practical concurrency controls. Both mechanisms have been reviewed and shown to be special cases of a more general concurrency control algorithm. This concurrency control algorithm has been described and proven to work correctly (i.e., produces only serializable schedules). Two-phase locking and timestamp ordering represent the two end points of a series of concurrency controls. Each results from choosing a different value for the strictness level and is considered to be a special case of the general concurrency control described in this paper.

REFERENCES

1. Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger. "The Notions of Consistency and Predicate Locks in a Database System." *Communications of the ACM,* 19 (1976), pp. 624–633.
2. Kung, H. T., and J. T. Robinson. "On Optimistic Methods for Concurrency Control." *ACM TODS,* 6 (1981), pp. 213–226.
3. Lamport, L. "Towards a Theory of Correctness for Multi-user Database Systems." Massachusetts Computer Associates/Report CA-7610-0712, October 1976.
4. Reed, D. P. "Naming and Synchronization in Decentralized Computer Systems." MIT/LCS/Report 205, September 1978.
5. Rosenkrantz, D., R. Stearns, and P. Lewis. "System Level Concurrency Control for Distributed Database Systems." *ACM TODS,* 3 (1978) pp. 178–198.
6. Bayer, R., H. Heller, and A. Reiser. "Parallelism and Recovery in Database Systems." *ACM TODS,* 5 (1980) pp. 139–156.
7. Gray, J. N. "Notes on Database Operating Systems." in G. Goose and J. Hartmanis (eds.), *Lecture Notes in Computer Science.* New York: Springer-Verlag, 1978.
8. Papadimitriou, C. H. "The Serializability of Database Updates." *JACM,* 26 (1979) pp. 631–653.
9. Holt, R. "Some Deadlock Properties of Computer Systems." *ACM Computing Survey,* 4 (1972) pp. 179–195.
10. Bernstein, P. A., and N. Goodman. "Timestamp Based Algorithm for Concurrency Control Problem for Multiple Copy Databases." *IEEE Proceedings of the International Conference on Very Large Databases,* (6), 1980, pp. 285–300.
11. Thomas, R. H. "A Solution to the Concurrency Control Problem for Multiple Copy Databases." *IEEE Proceedings of the Computer Society International Conference* (16), 1978, pp. 56–62.
12. Bernstein, P. A., and N. Goodman. "On Concurrency Control in Distributed Database Systems." *ACM Computing Survey,* 13 (1981), 185–221.

Crypto-secure operating systems

by GEORGE I. DAVIDA and BRIAN J. MATT
University of Wisconsin—Milwaukee
Milwaukee, Wisconsin

ABSTRACT

Enforcement of security policies in most computer systems is carried out by a centralized mechanism, usually the operating system kernel. Such mechanisms strip users of having complete control over access to their property. This paper describes a system that allows the user to determine the security policy to be enforced by independent, secure processes called Guardians. The system provides some of the tools that the Guardians can employ, and the proposed mechanisms provide more flexible security environments for users. In addition, penetration of part of the system does not jeopardize the security of the whole because resources are not owned entirely by one "super user." The mechanisms rely heavily on encryption, which mediates access to objects in the system. The role of the system is reduced to that of a messenger, and although it is possible to deny service, the security of the system is not compromised.

INTRODUCTION

Protection of programs and data in a computer system is the task of the operating system, and security policies evolve over time as multiuser systems change. Because users must share software and devices, including the processor and mass storage, the task of managing the resources is given to the kernel. This leads to the user's total dependence on the operating system for enforcing security policies. The operating system, in turn, depends on some hardware support to protect itself and the user against malicious or accidental damage. However, most of the protection policies are enforced by software. This is because the operating system is responsible for considerably more than protection. It also performs mundane tasks for the user because it is centralized. Centralization has led to the user's dependence on the operating system for a broad range of services. This is best exemplified by the UNIX operating system.[1] An exception to this is the UCLA kernel.[2]

SECURITY MODEL

In this paper, we propose to shift responsibility for enforcement of security from the kernel to the user. The operating system will be called on to perform communication tasks between processes, but it will not be responsible for the security of these messages. We also do not wish to rely on the system's reliability to protect against unauthorized access. Failure may result in the denial of service, but not the breach of security.[3]

The protection-related hardware of a computer system provides structural support to the operating system. This consists of memory management and user–supervisor partitioning of the instruction repertoire. These mechanisms are insufficient to enforce a fine-grained protection policy. They are similar to the walls of a building: A building's security does not depend on its walls, but on limiting access to individual rooms through such mechanisms as keys, guards, and the like. We propose to provide a similar environment, where the hardware provides the basic underpinnings, and the rest is done by the user—the owner of the objects—using new mechanisms. The environment in which this occurs should be a decentralized system where decisions about security are made locally by the user.

PROPOSED SYSTEM

Our system is a modification of the Berkeley 4.2 UNIX system, which, by virtue of its sophisticated interprocess communication facilities, provides a good basis for developing message-based systems. Our design consists of the following elements: better process protection, especially of any secret keys a process may possess; secure communiction paths between users and processes; independent processes (called Guardians) that act as surrogates for the owners of the system's resources; and a file system that supports these facilities. These elements result in a system where the kernel's role in supporting security is minimized, with a resulting reduction in dependence on kernel reliability. The Guardians are similar to the Monitors described by Graham and Denning.[4,5]

PROCESS PROTECTION

In time-shared systems the processes inhabit the same memory and mechanisms for this must exist while security is maintained. In a single-processor system, one process cannot be interfered with by any other process as it runs. When a process is inactive it is vulnerable both to the reading of sensitive information, such as security keys, and to alterations of its environment. Ideally, there should be hardware mechanisms by which sensitive areas of a process are protected via encryption, when a process is suspended. Because systems that are now available lack such a mechanism, we emulate this feature in our software. When a process is to be swapped out, the period of time that it may be exposed as a result of inactivity is considerably greater than that of a time slice. Therefore the degree of protection afforded to the process in this instance is considerably greater.

USER COMMUNICATION

The role of the system is that of a communication switch that routes messages among the processes and users of the system. The system should not be expected to provide secure communication—that responsibility should rest with the users and the processes. The processes can obtain secure communication in a system using cryptographic protocols implemented with software or hardware. If their local equipment has the capability, the users can do the same. However, if the user lacks this capability, then the system must supply it. Therefore either a user or a process can request that the system cryptographically seal the communication path between them. The seal is provided currently at the interface between the device driver and the device controller via the interrupt-handling routines.

FILE STRUCTURE

A UNIX file system can be viewed at two levels. At the first level, which is more primitive, the file system is flat. It consists of such data structures as a boot block, super blocks, and data blocks.[1] At the second level, the system is tree-structured, with directory files forming the nodes and the regular files and certain special files forming the leaves. All files consist of a single i-node located at a fixed position on a disk, zero or more data blocks, and possibly indirection blocks. The file's i-node contains the protection bits, ownership information, certain time stamps, and pointers to data blocks and indirection blocks.[1]

In our system there is an additional protection bit used to denote a secure file, and the following additional fields have been included:

1. Security flag field. The bit flags that comprise this field among other things control, in a secure file, whether the system standard Guardian or some other user-specified Guardian is to be used. Other fields determine whether the first block of the secure file is a secure header block if there is a control file associated with this secure file.
2. Security function control field. This field determines whether a Guardian should be invoked when a particular action occurs.
3. Device numbers. This field provides the major and minor device numbers for the Guardian, assuming the system standard Guardian is not assigned, and the control file if one is in use.
4. Guardian i-number and control file i-number. This field is similar to the device numbers field except that i-numbers are stored.
5. Consistency fields.[2] Two of these fields are present. They contain a cryptographic hash of the i-node fields, the indirect blocks, and the data blocks.
6. Reference count. If the file is a Guardian or a control file, this field contains a count of the number of secure files that employ this file.

These additions to the i-nodes result in connections between files other than those of the logical file system. The connection between a secure file and its Guardian or control file is a direct link. The direct link allows the system to locate the Guardian or control file without performing time-consuming path traversals. Indeed, a path to the Guardian or control file may no longer exist in the logical file system, yet the direct link still exists. The lack of a pathname for the Guardian of control file would result in a lack of direct access to these objects through the logical file system. The cross-device nature (both logical and physical) of a direct link allows for different physical protection strategies to be used for the Guardian and the control file. The direct link for a file may denote the file itself, allowing a control file to contain its own access permissions or a Guardian to guard itself. This answers the old question, "Who will guard the guards?"

The consistency filed contains an encryption of the fields or the i-node in addition to whatever else (indirection blocks or data blocks, for example) the user may specify. The encryption is by a user-supplied key and the result may be stored in this field, displayed to the user, or both.

GUARDIANS

It is useful to move security-relevant functions out of the kernel.[3] Our system uses processes that achieve the twin benefits of distribution and kernel simplification. The system is based on the concept of a Guardian. A Guardian is a process that acts as a surrogate for the owner of a file system object. This concept is appropriate for multiuser computer systems because the owner of an object usually has partial, if not complete, control over the level of "protection" afforded an object. It is also approprite to keep the function of the operating system kernel to a minimum.[6] Our system removes security decisions from the kernel and uses encryption to limit our dependence on the kernel's reliability. The kernel provides services to the Guardians and implements their decisions. It also provides communication paths between processes, but it is not responsible for securing the channel. Secure communication protocols furnish Guardian-to-Guardian communication.

Guardians use dynamic profiles of users and fingerprints of processes, kernels, and file system objects to establish the identity of users, processes, systems (kernels), and file system objects. This is an absolutely essential feature. A major advantage of our system is that there is no need for a super user. Each system object, terminal line, and printer spool has a separate owner. In certain cases the owner of a file does not exist and the file can be modified only when the system is off line. Because each Guardian is responsible for those objects entrusted to its care, each enforces exclusive use of secure objects. This solves the problem of interruptability.[7] Guardians enforce policies that the owners of system objects deem appropriate by using encrypted tokens, time stamped tickets, or records stored in secured files. When a Guardian is running, system operations are those initiated by its actions, except for interrupt handlers. When a Guardian or any other process is suspended it loses that control.

A Guardian is an independent process in that it is not subject to normal restrictions placed on signals and process termination in a UNIX process. A Guardian is created when a secure file is accessed during the course of a system call when the system converts to serve more than one user. The term accessed in this case refers to the obtaining of an i-node table entry during pathname resolution and subsequent use within that system call.

If access to a secure directory is required during pathname resolution, a Guardian is invoked. The actual file that is the target of the system call (if it is a secure file) also will have its Guardian invoked. If the system call "returns" a file table pointer, the associated file descriptor returned is a socket that connects the calling process to the Guardian (if any access is allowed), and the Guardian has direct access to the secure file via its own file descriptor. Subsequent reads and writes from the calling processes are mediated by the Guardian. Only the Guardian may have a file descriptor that actually refers to a secure file. If, after the pathname search the system call does not return a file descriptor, the caller goes to sleep and the secure file's Guardian runs. The Guardian makes the determination of whether and how the system call should proceed and then exits, returning its decision to the system. At this point the caller wakes up and the system call proceeds as directed by the Guardian. When a system call is suspended pending the decision of a Guardian an alarm is set before the caller goes to sleep. This alarm will prevent the caller from waiting indefinitely. If the Guardian has not rendered a verdict in the time specified by the caller, the decision is assumed to be complete denial of service. The Guardian is notified and the caller awakes. This mechanism allows users to produce custom-made audit trails via Guardians.

Once a Guardian is created, it runs as the owner of a file. It does not change its identity during the course of execution. With Guardians available there is no need for the standard UNIX set user and set group identification facilities, much less the change user facility. Normally a Guardian is not "execl-able," that is, the execl system call will not work and the calling process will not be overlaid with the Guardian. The only exceptions to this rule occur just prior to the system going into multiuser mode or when the Guardian of a terminal line gives up its post. When users log in, the post is surrendered to give each user a separate Guardian on the line for the remainder of the user's time on the system.

The system provides a number of aids that allow the Guardians to perform their tasks. Guardians must be able to verify their environment, control the execution of their children, and verify the consistency of a process and its text file. The system does not supply any additional process-to-process communiction facilities beyond what is available in UNIX 4.2 BSD. However, sockets have been modified so that a Guardian can support such operations as a seek on a socket (that is, connected to a Guardian that supports it). When the system goes into a multiuser state the "init" process is allowed to execl a number of Guardians. These Guardians protect critical devices that are accessed by means other than the file system (terminal lines via terminal keyboards) or are considered so important that a Guardian should always be on duty.

GUARDIANS IN ACTION

The following are some examples of Guardians in action.

Act 1

A Guardian is camped on a terminal line where no one is currently logged in. If the Guardian will allow a process to read or write to this line which may be connected to a modem or printer, a socket with the standard name of the line will be provided by the Guardian, case A. If communication to the line is not allowed, the configuration is that of case B.

$$\begin{array}{ll} D_{tty} : D_{dd} : G_{tty,utty} : & D_{tty} : D_{dd} : G_{tty,utty} \\ \text{case A} & \text{case B} \end{array}$$

D_{tty} The terminal, D_{dd} The device driver.
$G_{tty,utty}$ The Guardian of file tty owned by utty,
: A communication path.

The Guardian in case A has full responsibility for protecting the line from processes in the system on behalf of the line's owner, who may not exist.

Act 2

The terminal line supports logging in from some remote device, a terminal or another processing unit.

$U_a : D_{tty} : D_{dd} : G_{tty,utty}$ case A
U_a The user a.

If the remote device, depicted as a terminal, is going to send or receive critical data, the configuration is first changed to:

U_a: $D_{tty} :_e D_{dd} :_e G_{tty,utty}$ case B
$:_e$ A system supported cryptographic communication path.

It is possible the user may provide his own encryption

$U_a :_e D_{tty} :_e D_{dd} :_e G_{tty,utty}$ case C

In case B, the encryption of incoming data is performed by the routine that services a data-received interrupt. The data are transformed just prior to being written to the device register (memory mapped machine).

Act 3

The terminal line Guardian satisfies itself as to the identity of the user (by comparison with a dynamic user profile) and determines that the user should be allowed to use this line. The terminal line Guardian is replaced by a Guardian of the user's choice.

$$\begin{array}{llll} U_a & :D_{tty} & :D_{dd} & :G_{tty,utty} \\ & & & \quad | \\ & & & \; F_{Prof} \end{array}$$

$|$
F_{Prof} The Guardian has opened the file F_{Prof} where Prof represents the contents of the file.

When the terminal line Guardian is created, it is given access to files that contain the profile information used to verify the identify of the user. This profile information is not necessarily independent of environmental factors such as the time of day, which terminal line is involved, or any other condition the Guardian can test. If the Guardian is satisfied, it arranges for the creation of a Guardian for the user. The line Guardian execls the user Guardian (which itself is guarded) and is replaced by it:

$U_a :D_{tty} :D_{dd} :G_{tty,utty} \Rightarrow \text{execl } F^S\{G_{user,a}\}_a$
$\Rightarrow$execl The initiation of the system call execl.
$F^S\{G_{user,a}\}$ The secure file that contains $G_{user,a}$, and is owned by a.

If permission to execl $F\{G_{user,a}\}$ is granted, the following state is reached:

$U_a :D_{tty} :D_{dd} :G_{user,a}$

At this point, or at any time after, if the user wishes to receive communications from other processes directly he may instruct the user Guardian to put out a socket:

$U_a :D_{tty} :D_{dd} :G_{user,a} :$

The Guardian, perhaps after soliciting advice from the user, determines whether it should allow some process to talk to the user and whether it should provide a filter against certain undesirable messages.

Act 4
The user obtains a command interrupter:

$$\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}}$$

The Guardian forks; its child uses verifiction techniques to check that the shell has not been altered "improperly":

$$\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a} \rightarrow_{verf} F\{P_{csh}\}}$$

If $\mathrm{G_{user,a}}$ is satisfied it performs the fork and execl:

$$\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a} : P_{csh}}$$

Act 5
The user, having successfully logged in, wishes to alter some especially critical information, for example, his password. A file containing such critical data is a secure file, so the following events ensue:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} :_e G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{passwd} \Rightarrow_{open} F^S\{_{passwd}\}_{pd}}\\ \mathrm{U_a : D_{tty} : D_{dd} :_e G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{passwd} : G_{passwd,pd} : F^S\{_{passwd}\}_{pd}}\end{aligned}$$

$\mathrm{P_{csh}}$ A process executing csh.
$\langle \mathrm{P_{csh}} \rangle$ The process $\mathrm{P_{csh}}$ in a dormant state.

At this point $\mathrm{G_{user,a}}$ uses an ioctl or a message to the user to cryptographically seal as much of the communication channel as possible, given available hardware, and then opens the password file. The two Guardians use cryptographic protocols to set up a secure link between one another.

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} :_e G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{:_E P_{passwd} :_E G_{passwd,pd} : F^S\{_{passwd}\}_{pd}}\end{aligned}$$

$:_E$ A Guardian supported cryptographic communication path.

The file Guardian would write encrypted versions of the password (for example, the password scheme of UNIX, create a checksum for the record, and send this data through to the file). It is always possible that the file Guardian may request additional user verification before updating the file.

Act 6
The user wishes to use a screen editor (called vi) on a secure file:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{csh} \Rightarrow_{execl} F\{P_{vi}\}}\end{aligned}$$

$\mathrm{F\{P_{vi}\}}$ is not a secure file, so execution proceeds normally (subject to any verification):

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{vi} \Rightarrow_{open} F^S\{data\}_b}\end{aligned}$$

The $\mathrm{P_{vi}}$ process attempts to open the secure file. The file's Guardian is invoked. The new Guardian informs $\mathrm{G_{user,a}}$ what $\mathrm{P_{vi}}$ is doing. $\mathrm{G_{user,a}}$ acknowledges, and in so doing assumes responsibility for $\mathrm{P_{vi}}$'s actions:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{vi} : G_{data,b} : F^S\{data\}_b}\end{aligned}$$

$\mathrm{P_{vi}}$ makes a temporary copy of $\mathrm{F^S\{data\}_b}$ under a secure directory, uses cryptographic read, and writes:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{vi} : G_{data,b} : F^S\{data\}_b}\\ &\qquad\qquad\mathrm{: F\{temp\}_b}\end{aligned}$$

Then $\mathrm{P_{vi}}$ closes $\mathrm{F^S\{data\}_b}$:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{vi} : F\{temp\}_b}\end{aligned}$$

When the data are written back to $\mathrm{F^S\{data\}_b}$ a similar set of operations is performed. In order for $\mathrm{G_{user,a}}$ to perform its supervisory function the kernel provides it with some control over its descendants. It can have the kernel seek the Guardian's permission prior to performing selected system calls (specifically opens).

Act 7
The user wants to read his mail and will send a reply. The mail program is called and it in turn opens user a's mail file:

$$\begin{aligned}\mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{chs}} \rangle\\ &\mathrm{: P_{csh} \Rightarrow_{execl} F\{P_{mail}\}}\\ \mathrm{U_a : D_{tty} : D_{dd} : G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{mail} \Rightarrow_{open} F^S\{a\text{'s mailfile}\}_a}\end{aligned}$$

Replace $\mathrm{U_a : D_{tty} : D_{dd}}$ by $\{\mathrm{a} \ldots \mathrm{dd}\}$ and let user a's mailman Guardian be invoked. This Guardian verifies user a's identity and then proceeds:

$$\begin{aligned}\{\mathrm{a} \ldots \mathrm{dd}\} \mathrm{: G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{mail} : G_{mail(a),a} : F^S\{a\text{'s mailfile}\}_a}\end{aligned}$$

At this point user a can read the mail. Note that if there is something dangerous about the mail that user a is about to read, the line Guardian is there to protect the user. If additional security is required, communication paths can be sealed and the mail message itself can be encrypted. Now user a would like to reply to the message by sending a letter to user b. The mail program opens user b's mail file (not shown) and user b's mailman is invoked. This new Guardian verifies the sender and screens the incoming mail for letter bombs:

$$\begin{aligned}\{\mathrm{a} \ldots \mathrm{dd}\} \mathrm{: G_{user,a}} &: \langle \mathrm{P_{csh}} \rangle\\ &\mathrm{: P_{mail} : G_{mail(a),a}} &&\mathrm{: F^S\{a\text{'s mailfile}\}_a}\\ & &&\mathrm{: G_{mail(b),b}}\\ & &&\mathrm{: F^S\{b\text{'s mailfile}\}_b}\end{aligned}$$

KERNEL MODIFICATIONS

The kernel in this system has been modified to provide support to the Guardians and not to determine policy. The Guardian support facilities include the following:

1. Creation and termination: When a Guardian is created, unlike the case with "normal" UNIX processes, it has no parent. The Guardian does not terminate on externally generated signals, and its initial set of open files is con-

trolled by the data in the i-node of the file it is guarding as well as by the data in its own i-node.

2. Surveillance of children: A Guardian can arrange to "trap" selected system calls of its own descendants (but not those of other Guardians). The Guardian can thereby control the actions of its charges.
3. Verification of caller: The ability to identify the user of a communication line is essential to the proper functioning of a Guardian. Guardians can gain direct access to a terminal line and interact with the user for this purpose.
4. Verification of intended child at call (for execution), verification of a process during execution (frisking), and verification of system identity. The system employs a depository of "process" fingerprints under the control of a phantom user. A phantom user is one that can run or have non-Guardian processes on the system only when the system is in single-user mode. The depository is a secure file or collection of secure files.

HARDWARE SUPPORT

Current hardware configuration consists of a single MC68010 processor and peripherals. Proposed expansion is to include a second processor with local memory and special-purpose cryptographic hardware. The new processor will allow considerable improvements in performance as well as monitoring functions that are not practical on a single-CPU system. The new processor will implement such cryptographic tools as key sharing.[8] The National Bureau of Standards data encryption standard (implemented in special purpose hardware),[9] substitution permutation networks,[10] database security,[5,11] and arbitration of cryptographic protocols also will be added to the new system.

REMARKS

Most of the elements of security that we have presented can and should be implemented in hardware. In particular, process management must be implemented in hardware to make process suspension and activation safer.

REFERENCES

1. Ritchie, D. M., and K. Thompson. "The UNIX Time-Sharing System." *Communications of the ACM,* 17, (1974), 7, pp. 365–375.
2. Popek, G., M. Kampe, C. Kline, A. Stoughton, M. Urban, and E. J. Walton. "UCLA Secure UNIX." *AFIPS, Proceedings of the National Computer Conference* (Vol. 43), 1974, pp. 145–151.
3. Popek, G., and C. Kline. "Issues in Kernel Design." *AFIPS, Proceedings of the National Computer Conference* (Vol. 47), 1978, pp. 1079–1086.
4. Graham, G. S., and P. J. Denning. "Protection—Principles and Practice." *AFIPS, Proceedings of the Spring Joint Computer Conference* (Vol. 40), 1972, pp. 417–429.
5. Denning, D. *Cryptography and Data Security.* Reading, Mass.: Addison-Wesley, 1982.
6. Neumann, P. "Computer System Security Evaluation." *AFIPS, Proceedings of the National Computer Conference* (Vol. 47), 1978, pp. 1087–1095.
7. Ames, S., and J. G. Keeton-Williams. "Demonstrating Security for Trusted Applications on a Security Kernel Base." *IEEE 1982 Symposium on Security and Privacy.* Silver Spring, Md.: IEEE Computer Society Press, 1982, pp. 145–156.
8. Davida, G., R. A. DeMillo, and R. J. Lipton. "Protecting Shared Cryptographic Keys." *IEEE 1980 Symposium on Security and Privacy.* Silver Spring, Md.: IEEE Computer Society Press, 1980, pp. 100–102.
9. "Data Encryption Standard." FIPS Pub. 46, National Bureau of Standards, January 1977.
10. Kam, J., and G. I. Davida. "Structured Design of Substitution-Permutation Encryption Networks." *IEEE Transactions on Computers,* 28 (1979), 10, pp. 747–753.
11. Davida, G. I., D. Wells, and J. Kam. "A Database Encryption System with Subkeys." *ACM Transactions on Database Systems,* 16 (1981), 2, pp. 133–140.

Panel: Trends in computer systems technology

Chair:
JOHN L. BERG, *Sperry Corporation,* St. Paul, Minnesota
Members:
SHELDON A. BORKIN, *IBM Cambridge Scientific Center,* Cambridge, Massachusetts
ALFRED W. DiMARZIO, *General Electric,* Bridgeport, Connecticut
JOHN HEFNER, *National Bureau of Standards,* Washington, D.C.
EUGENE LOWENTHAL, *Microelectronics and Computer Technology Corp.,* Austin, Texas
REID SMITH, *Schlumberger-Doll Research,* Ridgefield, Connecticut

Can the innovations in computer technology over the next 10 years be predicted on the basis of the measured trends of the last 10 years? Some analysts say no, citing microcomputers/workstations, database/information resource management, LANs, artificial intelligence applications, and user interfaces as technologies for which extrapolations are not easily made. A panel of industry analysts discusses this issue.

Panel: Visual languages

Chair:
S. K. CHANG, *Illinois Institute of Technology,* Chicago, Illinois
Members:
ADARSH ARORA, *Gould Research Center,* Rolling Meadows, Illinois
ROBERT B. GRAFTON, *Office of Naval Research,* Arlington, Virginia
MARGARET A. KORFHAGE, *Trammel Crow Company,* Dallas, Texas
MOSHE M. ZLOOF, *M. M. Zloof, Inc.,* Dobbs Ferry, New York

This session focuses on the theme of visual langauges. Topics to be considered by the panelists include visual programming, form-oriented programming visual interfaces for information systems, and the design of visual and icon languages. Visual languages will grow in popularity because of their emphasis on high information density, man-machine communication. The panel also considers the future of such languages.

Panel: Software engineering directives

Chair:
MARIO BARBACCI, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
Members:
JOSEPH BATZ, *The Pentagon,* Washington, D.C.
NICO HABERMAN, *Carnegie-Mellon University,* Pittsburgh, Pennsylvania
W. MacDONALD MURRAY, *General Dynamics,* St. Louis, Missouri

The Software Engineering Institute (SEI) is being established by Carnegie-Mellon University under the sponsorship of the Department of Defense to address the problems of software development. The overall objective of SEI is to increase the quality and decrease the cost of software by improving the software development process. As part of its mission, SEI will endeavor to expedite the transition of new software development technology from research into actual practice.

In this panel session, members of SEI and representatives from government and industry describe the goals of the institute, its organization, and the initial set of projects. Further discussion focuses on directions for software engineering research as envisioned by SEI.

Panel: Can we make programming teams succeed?

Chair:
ANNELIESE von MAYRHAUSER, *Illinois Institute of Technology,* Chicago, Illinois
Members:
J. DANIEL COUGER, *University of Colorado,* Colorado Springs, Colorado
NICHOLAS MARSELOS, *AT&T Network Systems,* Lisle, Illinois
CARMA L. McCLURE, *McClure Associates,* Chicago, Illinois

Almost since the beginning of programming, the problems programmers were involved with were complex enough that they had to organize and work in groups. Formally defined programming teams promised a more structured form of organization and greater project success. The high expectations these concepts generated were shattered when problems with software development persisted in spite of these new structures and technologically knowledgeable team members.

This panel discusses experiences with team concepts, analyzes them, and develops reasons why the application of team concepts may have failed, as well as suggestions on how to prevent such failure. In particular, motivational theory, task design, and team-building strategies are discussed.

Panel: Decision support systems

Chair:
ROBERT W. BLANNING, *Vanderbilt University,* Nashville, Tennessee
Members:
GARY W. DICKSON, *University of Minnesota,* Minneapolis, Minnesota
JOHN C. HENDERSON, *Massachusetts Institute of Technology,* Cambridge, Massachusetts
MILES H. KENNEDY, *Case Western Reserve University,* Cleveland, Ohio
BENN R. KONSYNSKI, *University of Arizona,* Tucson, Arizona
ANDREW B. WHINSTON, *Purdue University,* West Lafayette, Indiana

This panel discussion focuses on a broad range of topics concerning DSS. Included in the discussion will be

—The state of the art of DSS, including applications and organizational and technological issues affecting DSS development
—Current and projected research issues in DSS, including the potential impact on end users
—The impact of forthcoming technologies—such as more powerful microcomputers and executive workstations, inexpensive AI (LISP and PROLOG) processors, and improved data communication services—on the future of DSS

Panel: Information system support for engineering applications

Chair:
PETER DADAM, *IBM Heidelberg Science Center,* Heidelberg, West Germany
Members:
ALEX BUCHMANN, *National University of Mexico,* Mexico City, Mexico
KLAUS DITTRICH, *University of Karlsruhe,* Karlsruhe, West Germany
RANDY JOHNSON, *Boeing Computer Service,* Seattle, Washington
STANLEY SU, *University of Florida,* Gainesville, Florida

The use of the computer to aid engineering design such as CAD/CAM and robotics has flourished in recent years. However, such design applications have been handled as separate processes. Data generated from one application are not easily communicable to other processes. With the right enviornment, this need not be the case. In particular, integrated information systems supporting the entire process, from design to manufacturing to marketing, can be developed. Naturally, techniques and knowledge developed from database management systems can be of use.

This panel addresses the problem of defining an information system to support engineering applications. The state of the art in this area is briefly discussed. Further, the panel considers the problems and requirements of such a system as well as some of the proposed solutions.

Panel: Software engineering standards—today and tomorrow

Chair:
LAUREL V. KALEDA, *IBM,* San Jose, California
Members:
H. JACK BARNARD, *AT&T Information Systems,* Denver, Colorado
DAVID CARGO, *Honeywell Inc.,* Minneapolis, Minnesota
JOHN HORCH, *Teledyne Brown Engineering,* Huntsville, Alabama
PHILIP C. MARRIOTT, *NCR Corporation,* Dayton, Ohio
GEORGE D. TICE, JR., *Tektronix, Inc.,* Wilsonville, Oregon

Current software engineering standards activities are described, especially those sponsored by the IEEE Computer Society. Discussion by the panelists focuses on such topics as

—Standards, guides, and practices under development and already approved
—The development process used for IEEE-sponsored standards
—The interrelationships of the IEEE Standards/Practices/Guides and other software engineering standards, such as Mil-S-52779A SQA program requirements
—Future directions of these efforts

NETWORKING

HARVEY FREEMAN, Track Chair
Architecture Technology Corporation
Minnetonka, Minnesota

SNA directions—a 1985 perspective

by R. J. SUNDSTROM, J. B. STATON, G. D. SCHULTZ, M. L. HESS,
G. A. DEATON, L. J. COLE, and R. M. AMY
IBM Corporation
Research Triangle Park, North Carolina

ABSTRACT

Since its announcement in 1974, SNA has evolved in terms of its functional content, configurational flexibility, and network management services. This paper briefly traces this progress to the present, and examines the more recent advances in greater detail. We then discuss known requirements for enhanced application and transaction services, for additional provisions for very large networks, for continuing exploitation of small-system and transmission media advances, for inclusion of additional management capabilities, and for further accommodation of network standards, all of which will shape future SNA developments.

INTRODUCTION

IBM's commitment to maintain and extend Systems Network Architecture (SNA) as its blueprint for the design of its communication products and for interconnecting them within networks has been reaffirmed and widely publicized over the years. From time to time, a checkpoint is useful to see where we have been and where we are going with SNA. One such checkpoint, given in 1980,[1] traced the progress of SNA, both as an architecture and as a set of products, over its first six years. Today, a year after IBM has marked the tenth anniversary of SNA's introduction, it seems appropriate again to examine SNA's past and current status, and to consider its future. Because the family of SNA products has mushroomed to such an extent (both inside and outside IBM), a description of implementations would well merit a paper of its own; but we confine ourselves here primarily to the architectural aspects.

In looking at the innovations introduced by SNA, two conceptual notions independent of the functions provided stand out; both were borrowed from other disciplines within computer science, but were freshly applied together to the networking context.

The first of these was the basic notion of the *network architecture* itself. This notion, originating in computer design,[2] resulted in the superimposition of a logical network on the underlying physical network. Thus, the interface presented to the user by the logical network could hide the actual underlying physical realities and provide additional functions not provided at the rudimentary physical level.

The second notion, originating in operating systems practice,[3] was to structure functional offerings in a *layered* fashion. Together, these notions resulted in SNA being structured as a set of layered logical networks where the outermost network offers an interface for the end user, and each successive inner network offers distinct functions for the next higher layered network in a well-defined fashion. Changes to one layer need affect a higher layer only to the extent that new function is offered, not because old functions are achieved internally in a different way. The notion of layering continues to be fundamental to the orderly evolution of SNA.

In the next section, we review briefly the highlights of SNA's evolution since its introduction. In the ensuing sections, we discuss the recent advances in more detail, according to function. In each case, we examine trends and speculate on the future based on new requirements on SNA.

EVOLUTION OF SNA—HIGHLIGHTS AND TRENDS

In 1974, SNA began as a simple tree-oriented, single-host network. Today, it has evolved to support multiple, independent, mesh-configured networks separately administered, but interconnected by gateways into composite networks. End users can freely communicate with application programs anywhere in these composite networks without being aware of the network configurations. Some of the major highlights in the progress of SNA are shown in Figure 1.

When SNA was introduced in 1974, it was supported by a single operating system (DOS/VS) running on a single host connected to terminals through a front-end communication controller. At that time, the only SNA terminals available were on the IBM 3600 banking system. Host programs communicated with the 3600 controller.

Within a year, SNA coverage was expanded to include remote communication controllers, the MVS environment, and additional supermarket and retail point-of-sale controllers and their terminals. Still, SNA remained a leased-line system only—a situation that changed in 1976 with the addition of dial capability.

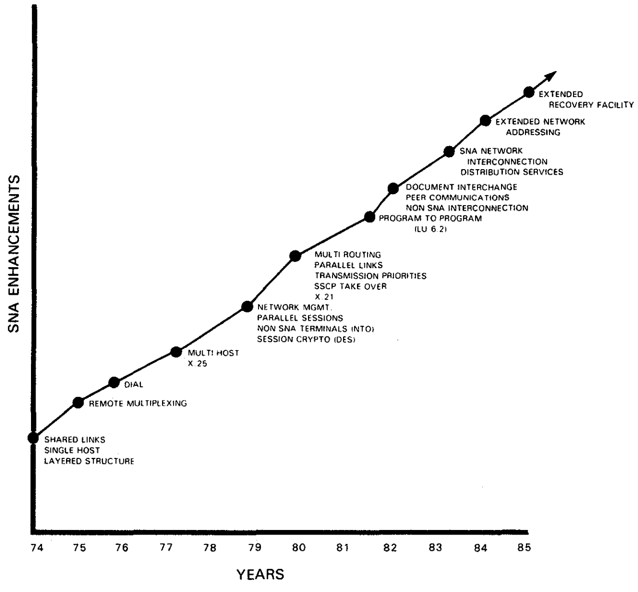


Figure 1—SNA evolution

In 1977, with the release of IBM's Advanced Communication Function (ACF), more general networking function became available. SNA allowed multiple host "trees" to be interconnected using single links to connect front-end communication controllers. This capability opened the way for a terminal to access application programs in multiple host processors. Also in 1977, IBM introduced its initial support for the X.25 standard for public packet-switched networks. We discuss this in greater detail in a later section.

With the basic networking in place, it was time for more emphasis to be placed on the incorporation of network management services. This focus, which began in 1979, continues to the present. The need for such services is proportional to the complexity of network configurations and how critical network reliability, availability, and serviceability are to the users of the network. With the increasing reliance by businesses on their network operations and with the widening scope of their applications, network management services have taken on greater importance as the architecture and implementing product set have been extended.

In 1979, IBM also enhanced session capabilities to allow parallel sessions between two application subsystems such as CICS and IMS. Another new session feature was the support of the NBS Data Encryption Standard (DES) for session cryptography. In addition, SNA capability to handle non-SNA terminals was significantly advanced by the inclusion of the NTO software product on NCP in the 3705 communication controller.

In 1980, significant extensions were made to SNA configuration flexibility and to its transport services. Improvements included multi-routing, parallel links between nodes, priority transport, and global congestion control within the network; fully meshed connectivity within the backbone transport network was introduced.

The 1980s have brought several significant advances.

1. Advanced Program-to-Program Communication (APPC), or LU 6.2, and node-type 2.1 introduced new and more general session and peer-oriented capabilities.
2. Document Interchange Architecture (DIA) and Document Content Architecture extended SNA support for office-systems applications.
3. SNA Distribution Services (SNADS) provided a new store-and-forward, or asynchronous, distribution service to SNA that complements the synchronous delivery support of sessions between two end users.
4. SNA Network Interconnection (SNI) and extended network addressing (ENA) enhanced SNA routing and configuration flexibility, particularly for large networks.

These more recent advances will be discussed in more detail in the following sections.

In summary, some of the major evolutionary trends in SNA have been

1. Increasing configuration flexibility, particularly to exploit advances in transmission technology;
2. A burgeoning set of products;
3. Greater attention to network management services;
4. Inclusion of network standards, as these have become available;
5. Widening support for non-SNA devices;
6. Expansion of routing and transport services to keep pace with installation of ever larger networks; and
7. Increasing function available to end users.

The steady development of SNA has been tempered all along by a concern for compatibility of past products with new SNA releases. The care taken is manifested by the continued operation of the original SNA terminals under the most recent releases of SNA. This migration sensitivity is one of the hallmarks of SNA evolution.

In the following sections, we discuss how new requirements will likely bear on the above trends. Some emerging trends will also be discussed; these concern an increased focus on continuous (24-hour) operation with a reduction in system-definition down-time, a greater emphasis on peer communication independent of the traditional backbone network, and exploitation of local-area networking and other state-of-the-art technology.

VTAM-NCP TRANSPORT NETWORK

In our survey of the directions that will shape tomorrow's SNA, we start with the VTAM and NCP transport network. VTAM and NCP are two of the first three SNA products (along with the 3600 banking system); they provide the SNA transport network and many of the control and service functions needed to operate SNA networks.

Recent additions to the VTAM-NCP transport network include SNA Network Interconnection (SNI) and extended network addressing (ENA). SNI was announced in November 1983; it provides for the interconnection of autonomous SNA networks through gateways, which consist of specialized interconnection logic in VTAM and NCP. SNI is appropriate for inter-company communication, for companies experiencing mergers and acquisitions, and for situations where independence of company divisions is needed; it enables two or more networks to merge for the purpose of user communication, but to be independently managed and controlled.

To maintain the integrity of the component networks, constraints are built into the gateway to prevent one network from disrupting an adjacent network or gathering information it shouldn't have access to. Flow control prevents one network from flooding its neighbors with more traffic than they are prepared to handle. Names (e.g., of LUs, discussed later) and routing addresses are also independently assigned. To resolve potential conflicts in name assignment, Network Communications Control Facility (NCCF) provides an optional name-aliasing facility. The SNI extensions to SNA are described in detail in other works.[4,5]

Because each of the independent networks can use its full SNA address space, SNI can also be used to configure networks larger than would otherwise be possible. Each network can allocate a pool of addresses available as local aliases for destinations in other networks; dynamic assignment can result in using them as needed, thereby sharing the pool over a large number of destinations in other networks. Thus, SNI provides the network interconnection function, and at the same time, possible relief from the addressing constraints some customers were experiencing. This is why SNI was provided prior to a more direct solution to the addressing problem (which is our next topic).

As SNA networks gew in size, a requirement arose to extend the original 16-bit address space. These 16-bit addresses were partitioned into two pieces, a subarea address that identified the destination subarea node (containing VTAM or NCP), and an element address that was used by the destination subarea node for routing, for example, to the correct VTAM application program or to the intended terminal. In a particular SNA network, the subarea address can be chosen to be any size from one to eight bits; the remainder of the 16 bits is then used for the element address. In theory, over 64,000 destinations could be addressed; in practice, this could not be achieved because the subarea/element split has to be uniform throughout the network, and the optimum split varies from node to node.

In September, 1984, IBM announced extended network addressing (ENA), which provides for 23-bit addresses, thereby allowing over eight million destinations in a single SNA network. The subarea portion is fixed at eight bits (the previous maximum for subarea addresses), and the element portion is fixed at 15 bits (the previous maximum for element addresses). These sizes were chosen because they were already accommodated by the existing routing tables.

We view this extension as an interim step, and recognize the need to provide much larger subarea addresses. This next step will be relatively easy from an addressing perspective because space exists for 48-bit addresses in SNA formats, but it will raise more serious problems with the routing schemes. Under the current SNA routing implementation, generating and storing routing tables become increasingly difficult as the network grows in size. This brings us to the next topic, the requirement for dynamic routing in SNA.

Today, if you have a large SNA network and want to add to it, you must first decide where to locate the new addition and what links should connect it to the existing network. Then, usually with the help of an IBM program such as Routing Table Generator (RTG), you design the routes of the enlarged network. Once this work is completed, you load the new routing tables into the subarea nodes of the network through a system definition process. This procedure lends itself to very efficient routing, since all the routes are predefined to the network; but, because of definition time and complexity, it limits the size of the network that can be practically supported.

In addition to supporting very large networks, SNA has a requirement to make all networks easier to install and change. One of our long-term design directions is to reduce, and wherever possible to completely eliminate, system definition.

One potential solution is to have the network update itself through an exchange of node and link characteristics whenever a change occurs in any of these parameters. The network could then use the most current topology information to compute the best route between points at the time the route was requested. This route computation could also consider a class of service requested by the user; this is how the user specifies to the network whether he needs, for example, the least-cost route, the route with the least delay, or the route with the greatest bandwith.

This dynamic routing capability will have a number of beneficial effects on SNA networks. First, larger networks will be possible, because the difficulties with the current route generation process will be eliminated and intermediate nodes will have to store routing information only for currently active routes, not for all potential routes. Second, networks will be easier to install and change by reducing the workload now experienced to do coordinated system definition. Third, this will be an important step toward continuous operation, because it will no longer be necessary to bring the network down for the purpose of updating routing tables. Further discussions of dynamic routing in SNA networks can be found in other works.[6,7]

The need for continuous operation runs deep in SNA, and is worthy of more discussion here. A number of features currently available in SNA can be used to increase the availability of networks and to insulate its users from outages; these include the following:

1. Pause and retry logic in Synchronous Data Link Control (SDLC), which allows SDLC links to remain operational across periods of transient errors on the links.
2. Multi-link transmission groups, which allow bundling a number of SDLC links into a single logical link. The sender schedules data traffic for the first available link in a group and the receiver reorders received messages if necessary to maintain the FIFO property of the logical link. Individual links can be dynamically added or deleted from a transmission group without disrupting the ongoing flow of information; a transmission group fails only when the last operational link in the transmission group fails.
3. Multiple routes. Networks can be configured with multiple pre-defined routes so that if the route serving a sessions fails, that session can be re-established over an alternate route.
4. Parallel sessions. SNA currently allows multiple simultaneous (*parallel*) sessions to be established between host application subsystems such as IMS and CICS. These sessions can traverse different routes through the network, and where supported comprise a resource pool; when a transaction program needs to communicate with a partner program at another host, the first available session with the desired class of service can be assigned from that pool. Should one session fail, other sessions in the pool will continue to provide session connectivity with the partner subsystem.
5. Host control-point (SSCP) takeover, which provides protocols in the NCPs for detecting failure of controlling hosts and informing their backups.
6. Distributed processing. By moving application programs and data closer to the user, the user can often continue working uninterrupted by link or node failures in the communication network.

While SNA today provides numerous functions that can be used to configure highly available networks, further requirements exist in this area. For example, while SNA provides multiple routes, should a route fail, the sessions it carries are deactivated prior to possible reactivation over a backup route. A possible extension is to have the network perform this route switch without disrupting the sessions.

Highly available systems typically avoid being sensitive to the failure of a single component. To eliminate this sensitivity in SNA, it would be desirable to attach peripheral nodes to the transport network through multiple links to different NCPs. This capability exists today by making the peripheral node appear to the network as multiple nodes with different sets of destinations; comparable support is also necessary in order to provide optimal routing when peripheral nodes are connected to the network using X.25, local-area networks, or other facilities that provide connectivity between large numbers of nodes. In these situations, data traffic could be routed directly from the peripheral node by the intermediate, high-connectivity communication facility to the NCP closest to the destination.

To eliminate having a single point of failure, provision will also have to be made for backup application subsystems. However, just having a backup application subsystem ready to start taking over the moment the primary application subsystem fails will not always be sufficient. Some critical application programs can support thousands of simultaneous users. It could take a number of minutes for the backup application subsystem to re-establish and resynchronize all the user sessions. For these critical applications, a need exists to pre-establish the backup sessions and have them available for immediate use should the primary application subsystem fail. For IMS applications, this capability is planned for availability in 1986 using the recently announced Extended Recovery Facility (XRF).

SMALL SYSTEMS

Thus far, we have been focusing on the VTAM-NCP transport network. We now look at SNA requirements and directions from the perspective of the peripheral nodes of the SNA network. In the past, these peripheral nodes have predominantly been display terminals, printers, and remote job entry stations. With the steadily decreasing cost of mini- and microprocessors and storage, more and more of the peripheral nodes are small *systems*, such as personal computers, distributed processors, intelligent workstations, and office systems, which because of their more general nature, have greater connectivity requirements than traditional terminal devices.

One such requirement is more flexible session connectivity. Figure 2 shows the sessions connectivity in SNA today. Host application subsystems can have sessions with other host application subsystems and with outboard terminals and small systems. The host-to-host connections can employ parallel sessions, which can be used to increase transaction bandwidth (each session can serve one active transaction), to provide for a distinct class of service selection, and to improve performance and availability by fanning out traffic across different routes between the hosts. Peripheral nodes, on the other hand, are currently limited to a single session per LU (a user port discussed later), and that session must be with a host application subsystem. A requirement exists for small systems to enjoy the session connectivity that hosts enjoy today, namely the ability to use parallel sessions and to have direct session connectivity with any other destination in the network.

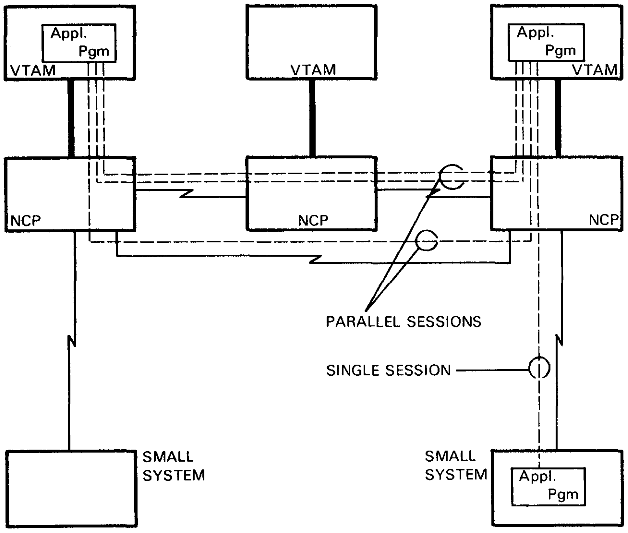


Figure 2—SNA session capability today

These small systems also have requirements for communication outside the VTAM-NCP environment. One simple but important form of communication is *direct peer-to-peer:* just two nodes and a link between them. While this is the simplest possible configuration, it is increasingly important because of the current trend in the communications environment towards high-connectivity multi-access facilities such as local-area networks, X.25, and ISDN, which are discussed later.

In 1983, a new peripheral node type, 2.1, was introduced into SNA to provide this peer-to-peer form of communication. Implementations of this new protocol are now available on the System/36, System/38, IBM 5520 Administrative System, Displaywriter, and Scanmaster. Direct peer-to-peer communication had been available earlier on IBM SNA products such as 8100/DPPX and previous releases of the 5520. The new node type 2.1 protocols provide the capability to carry LU 6.2 sessions (including parallel sessions), and provide the SNA direction for compatible, small-product, peer-to-peer communication.

In designing networking solutions for small systems, it is important to recognize the differences in operating environments between large and small systems. Procurement and operational decisions for small systems are generally decentralized and dynamic, resulting in frequent change. Yet, technical support from systems programmers and network operators is far more limited. Another difference affecting design decisions is that small systems typically need not support the high-traffic volumes of large systems; on the other hand, small systems have more stringent entry-cost requirements.

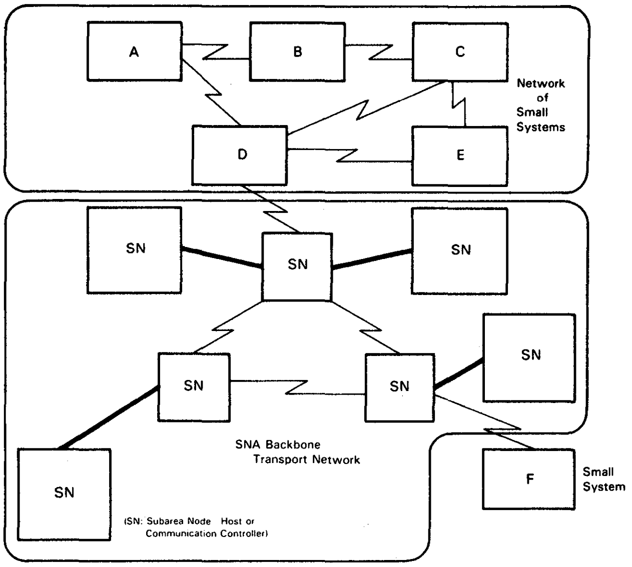

A
B
C
Network of Small Systems
D
E
SN
SN
SN
SN
SN
SN
SN
SNA Backbone Transport Network
F
Small System
(SN: Subarea Node Host or Communication Controller)

Figure 3—Combined network of small and large systems

We are currently investigating architectural solutions to meet the special needs of small-systems networking; further discussion on this topic can be found in the work of Baratz, et al.[8] Among the approaches being explored is a nonhierarchical, peer-to-peer scheme that uses dynamic routing protocols similar to those discussed earlier for the large-systems environment. This would support frequent network changes without the need for coordinated network definition of routes. Dynamic, distributed directories are also envisioned to avoid the need for coordinated network definition of the locations of network resources. This topic is discussed in more detail later.

The evolution of small-systems networking can hardly end with stand-alone networks; we anticipate a requirement to connect these networks to the high-capacity transport networks as shown in Figure 3. These connections should allow sessions between *small-system A*, for example, and an application in one of the S/370 hosts; they should also allow small systems to communicate peer-to-peer across the SNA backbone networks (for example, from *A* to *F*) and share the high-capacity links that are often in place.

Provisions are also needed for managing networks of small systems. These network management techniques should be consistent with the network management functions that are in place for SNA backbone networks so that when networks of small systems are connected to networks of large systems, the entire consolidated network can be centrally managed.

LOGICAL UNITS

Logical unit (LU) is one of the more abstract terms used in SNA, but it has a straightforward role, namely as the intermediary between the transport network and the people, devices, and application programs using or attached to the network. Figure 4 shows the position of the LU in the layered structure of SNA.

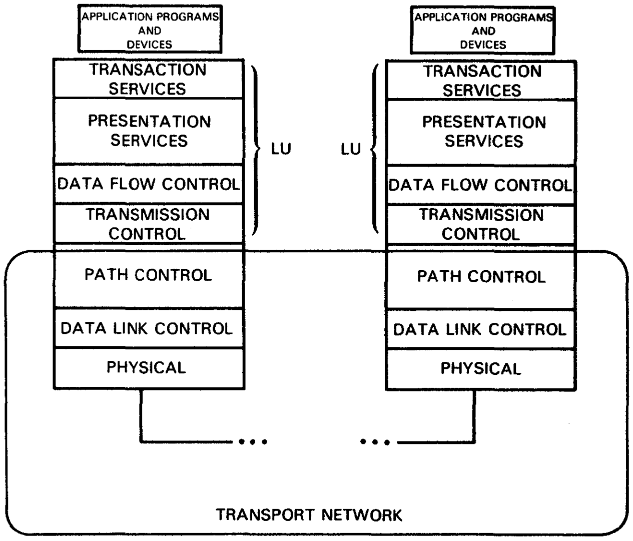


Figure 4—Role of the LU

To this point, we have discussed functions mainly in the transport network, which provides global protocols and services such as network-wide flow control and routing. By contrast, the LU is concerned primarily with session protocols for paired end users. LUs serve as the attachment points for the ultimate destinations of data (application programs, databases, and devices) and provide the end-to-end session protocols in support of communication between these network resources. While the transport network provides global flow control so that links and intermediate nodes in the network are not overloaded, the LU provides end-to-end flow control so that, for example, an application program does not send data to a printer faster than the printer can handle it. Other functions include session cryptography, name-to-address translation (using a distributed directory services capability of the network), and blocking and subdividing message units for network efficiency.

A number of LU types are defined in SNA, with the earlier ones (LU types 1, 2, and 3) optimized for asymmetric host-application-to-device communication. With today's trend toward personal computers, office workstations, and other small systems, new communication requirements exist at the LU level as well as at the transport network level. The key requirement at the LU level is for general program-to-program communication. A single set of protocols is needed for communication between all types of network nodes, including host-to-host, host-to-small-system, and small-system-to-small-system. It should provide a range of functions suited to such products as the IBM Personal Computer and the IBM 3820 Page Printer at the low end, and CICS and the System/38 at the high end. The protocols should also provide a new base for device support, allowing for ease in function distribution to the devices.

To meet these requirements, a new LU type, 6.2, was introduced in 1982 as Advanced Program-to-Program Communication. Initial IBM implementations included CICS, System/36, System/38, Displaywriter, Scanmaster, the IBM 5520 Administrative System, the IBM 3820 Page Printer, Print

Services Facility, and the IBM Local Network PrintManager on the IBM PC.

To support a broad range of distributed applications across any set of LU 6.2 products, LU 6.2 defines a set of generic commands (*verbs*) implemented by each LU 6.2 product in its own syntax, but with common semantics that application transaction programs can issue to communicate, independent of the details of the underlying configurations and protocols. The services provided by these verbs are defined in an IBM reference manual.[9]

LU 6.2 was designed to include SNA-defined transaction programs such as those for SNADS and DIA (discussed later) to serve device- or product-specific transaction programs such as those used to communicate with the IBM 3820 Page Printer, and to be used by user-written transaction programs. Users can provide their own transaction programs on LU 6.2 implementations such as CICS, System/36, and System/38, that provide for customer programmability. Such implementations are said to have an *open* application program interface (API); implementations in which the LU 6.2 implementation is limited to serving only prepackaged transaction programs are called *closed* API products.

To meet the low entry cost and the high function requirements, LU 6.2 has a base set of functions and a limited set of options. Every open-API implementation supports the base and may also support any of the option sets. One option implemented by CICS can synchronize updates to multiple databases so that all updates either succeed or fail together as an atomic unit of work. Closed API products need provide only the functions used by their coupled transaction programs.

LU 6.2 has recently been enhanced with the addition of option sets for transaction program security. The foundation of these protocols is a two-way verification exchange used to check the identity of the session partner. This is accomplished by having each LU generate and transmit a random number, and having the partner return that value encrypted under a shared session password. Later, when a transaction program initiates a conversation with a remote partner using the session, it can include a user ID and another password. Based on the trust established in the earlier verification, these fields need not be encrypted; they are used by the receiver to verify that the conversation initiator has the authority to gain access to the requested transaction program. Transaction programs with very high security requirements can additionally transmit fully-encrypted data throughout their sessions. Further information on LU 6.2 can be found in other works.[10–12]

In the following sections, we explore the progress in the transaction services layer, the top layer of the LU. The basic application interface of LU 6.2 is facilitating progress in several areas of transaction services.

SNA DISTRIBUTION SERVICES

SNA Distribution Services (SNADS), made available in 1984, consist of a set of IBM-supplied transaction programs that use a network of LU 6.2 sessions to provide a store-and-forward distribution capability in an SNA network. The LU 6.2 base provides a synchronous, connection-oriented, logically point-to-point service for application programs, analogous to that of a telephone service. SNADS builds on this basic service to provide a connectionless, noninteractive distribution service, analogous to that of a mail service.

This service allows users needing to distribute material to submit their request to a *distribution service unit* (DSU), a SNADS component, to initiate full delivery. It frees the users from further responsibility. A network of DSUs connected by LU 6.2 sessions handles the complete distribution process. SNADS manages the subsequent distribution, including queuing for resources, fanning out the distribution at the appropriate point in the network topology, and notifying users or their agent transaction programs at the destinations of the distribution's arrival.

SNADS is particularly appropriate for batch-oriented, one-way flows found in such applications as document distribution, file transfer, and job networking. By queuing for resources until they are available, SNADS can transport material through a network where facilities are not all simultaneously active. For example, portions of a network may be connected only by switched links that operate at certain times of day. SNADS frees the user of concerns about network resource availability.

A SNADS user is referred to by a two-part name, which is generally independent of the network topology; user location is determined by reference to a directory. SNADS allows this directory to be distributed in such a way that the complete directory database need not be replicated at every DSU. Incomplete directories have entries that point to DSUs where more complete information can be found.

SNADS allows up to 256 distinctions to be specified for any given distribution request. The user's material is transported in such a way that only one copy is sent to any distribution service unit in the path to a final destination. Copies are made only when necessary as paths to the multiple recipients diverge.

SNADS is currently implemented for a number of products that will use it primarily for document distribution; these are DISOSS/370, System/36, and 5520 Administrative System. IBM statements of direction exist for System/38, Series 1, and 8100. Because document distribution is the most common application in these first implementations of SNADS, the SNADS formats were designed to be compatible with those in Document Interchange Architecture (DIA), one of IBM's architectures for the office. Further information about SNADS may be found in other works.[13,14]

ARCHITECTURES FOR THE OFFICE

To handle situations generic to the automated office, IBM has developed two architectures specific to this important application, Document Interchange Architecture,[15–17] and Document Content Architecture.[17–19]

Document Interchange Architecture

Like SNADS, Document Interchange Architecture (DIA), introduced in 1982, is also part of the transaction services layer of SNA. It provides a set of protocols that define how several common office functions are performed cooperatively by IBM products. These include the filing, searching, and retrieving of documents and memos as part of DIA's document library services. DIA's document distribution services provide for the sending and receiving of documents or memos via SNADS or the basic LU 6.2 sessions, and include listing items pending receipt, canceling or resequencing their delivery at the recipient's request, and allowing access to software mail boxes and files by other authorized users. The formatting and processing of documents are defined in application processing services. The implementation status of DIA is similar to that of SNADS, but also includes IBM PC, Displaywriter, and Scanmaster implementations.

Document Content Architecture

A vital component of the office architectures is the Document Content Architecture, which provides the formats for describing the form and meaning of objects that are managed by DIA. Currently, two forms are implemented.

1. Final-form Text provides primitive format controls within a data stream in a generic fashion to allow presentation fidelity in a device-independent manner. This allows the sender of a document to control the formatting and print integrity of a document at its final destination without having to know the destination's print device characteristics.
2. Revisable-form Text allows interchange of draft documents in a form that is suitable for revision. Text processing indicators are included with the text, and may themselves be revised.

While the two current types of Document Content Architecture are a good beginning, others are needed to describe the mixing of information types within a document. Text, graphics, image, and voice annotation data are all amenable to the same design approach. A Document Content Architecture serving mixed data would allow documents that integrate a variety of data types to be exchanged by future office workstations.

DIRECTORY SERVICES

One of SNA's design goals has been to insulate the user and the user's application program from the characteristics of the communications network, and to allow users and applications to be moved among processors without impacting other users and applications that communicate with them.

To do this, SNA has distinguished between resource names and their addresses. A name is a relatively stable identifier that users and applications can apply to other users or programs. By contrast, an address can vary according to operations decisions made in the network. Users and their application programs access resources by name. The system uses the name as a key to a directory that provides the current address of the requested resource.

In early SNA networks, the directory tables were defined in the system services control point in a System/370 processor. The control point acts as a mediator in LU-LU session initiation, translating names to addresses and checking resource availability. The use of a central directory in the control point simplified the management of the directory for additions, deletions, and changes.

With the introduction of multiple-host networks in 1977, the control points cooperated to provide directory services, with each control point being responsible for the detailed address information on a subset of the pool of LUs in the network; each control point knew all LU names and the associated control point that could resolve a specific name to an address. Successive designs of the control point have reduced the amount of coordinated predefinition of resources by eliminating redundancy among the directories in different control points. In an SNI environment, a control point can perform a trial-and-error search of other control points for LUs not found in its own directory; this further reduces the number of control points that need to be updated when new LUs are brought on-line.

In SNADS, a user directory is referenced to determine the distribution service unit (address) of an intended recipient. SNADS products have implemented their directories, which are manually maintained, as part of the DSU, rather than in the control point.

Two trends stress the current design point of manually maintained directories; as networks become larger, the frequency of directory updating increases and the number of directories that must be consistently maintained grows; furthermore, the trend toward peer connectivity among small systems requires that small systems also maintain directories, whereas they may have formerly depended upon a large host directory. Both trends result in many more directories and increased update activity in a network, and point to a growing need for the directories to be automatically and dynamically maintained.

Resources such as application programs, files, and LUs need to be registered only at their local (home) directories, at a minimum. Automatic network-wide searches of the various directories eventually results in the finding of the resource if an active path to it is available. Replication of directory entries throughout the network can have several benefits:

1. Performance in finding resources can be improved;
2. Switched links can be activated in order to complete required synchronous connections; and
3. Asynchronous distributions can be forwarded as far as possible into the network when an active path to a destination is not available.

To meet the requirement for more dynamically maintained, distributed directories in an SNA network, new protocols will be needed for resource registration and network searching. These protocols will apply between directory users and their directory providers and among the directory providers themselves.

MANAGEMENT SERVICES

The previous sections of this paper discussed the advances in the functional richness and configuration flexibility allowed by SNA. Now SNA users can connect multiple SNA networks, small systems to large systems, and various other devices, all within the same composite network. The actual attachments may be over various transmission media (e.g., SDLC telecommunications links, S/370 channels, and X.25). These enhancements exact a price in that they make the network more complex to manage.

Management services, otherwise known as *network management* or *communications network management*, include the monitoring and controlling of the SNA network and its associated resources. IBM is integrating management services into SNA to address the management problems inherent in the ever-increasing complexity of the communication environment.

History

The early releases of SNA provided limited management services in VTAM, supplemented in 1977 by the Network Operator Support Program (NOSP). In 1979, in the next major release after multiple-host support was introduced, the Network Communications Control Facility (NCCF), replacing NOSP, and the Network Problem Determination Application (NPDA) were included. NCCF was created to aid in operator control of the multi-host environment; NPDA, to help manage problem determination for the many different products that could be attached. In 1982, Network Logical Data Manager (NLDM) was introduced to help manage the logical resources (such as sessions and virtual routes) in SNA.

The original functions performed by NCCF, NPDA, and NLDM were largely on a product-by-product basis rather than being general functions defined by SNA. With the continued enrichment of SNA, management services have become a more integral part of that enrichment and go beyond the early product-specific design.

Requirements

The network owner or the provider of network services must be given the tools and resources to provide the reliable, high-performing secure services that users require. The network owner must also be able to manage the network configuration, effect changes, and monitor the use of network resources. The network operator (programmed, or human) must be able to invoke these functions from a central site, or to distribute control of the functions. Of course, all the functions must be performed in a cost-effective manner.

Major Categories of Management Services

The requirements for managing an SNA network fall into four major management services categories.

1. *Problem management*—the function of managing a problem from its detection through its resolution. The steps of problem management are: (1) problem determination, (2) problem diagnosis, (3) problem bypass and recovery, (4) problem resolution, and (5) problem tracking and control.
2. *Performance and accounting management*—the process of quantifying, measuring, reporting, and controlling the usage, responsiveness, availability, and cost of a network.
3. *Change management*—the planning, control, and application of changes (additions, deletions, and modifications) to the resources of a network.
4. *Configuration management*—the control of information necessary to both logically and physically identify network resources, and to indicate their relationships to one another.

Management Services Components in SNA

The functions provided by NCCF, NPDA, and NLDM are represented in the architecture[20] by a management services component in the control point, physical unit,* and individual layers of SNA. This structure provides a framework by which the aforementioned requirements can be satisfied.

Each layer of SNA (see Figure 4) is responsible for controlling the resources associated with that layer. This is accomplished through a component called *local management services*. For example, routing information that is used for problem management is gathered by the local management services component in path control. Once gathered, this information is sent to the management services component in the PU.

Physical unit management services is responsible for gathering management services information local to its node or attached links, performing some services such as reformatting and time-stamping the data, and sending the information to the control point for processing. Control point management services is responsible for collecting management services data from the network, analyzing the data, and taking the appropriate action based on that analysis.

Current Management Services Functions

Alert

The Alert is a problem management function that is used to report a loss or impending loss of availability of a resource. Once an error is detected that requires corrective action by a network operator (human, or programmed), the Alert is sent to the control point. The Alert contains a general classification of the problem, a description of the cause of the condition, identification of the failing resource, and addi-

* A physical unit (PU) is an addressable entity like the SSCP or LU. One PU exists in each node for local control, and to represent the node to a control point (SSCP) in a host processor that is currently controlling the PU.

tional details pertaining to the problem. The management services component of the control point analyzes the data and reports the condition along with a recommended action to the network operator.

Problem determination statistics

Statistics are gathered by the data link control layer and are sent to the control point when a threshold is reached, or when solicited by the network operator. These statistics can be used as additional detail in problem management. The statistics include counters such as those for total transmit data, transmit errors, total receive data, receive errors, and polling information. For X.25, counters include statistics pertinent to the access links and virtual circuits.

Modem tests can be performed using the link problem determination aid (LPDA). This function allows the status of the modems and remote DTE interface and the results of the modem tests to be determined.

The link resource control function allows the network operator to dynamically change the threshold values of link statistics. It also allows the operator to disable the use of LPDA.

Response-time monitoring

Response-time monitoring (RTM) allows the network operator to validate certain predetermined end-user service levels for specified LU-LU sessions. Response time is measured by the time elapsed between LU recognition of an end-user request and reception by the LU of the resulting reply from the session-connected partner. The predefined values can be changed dynamically by the network operator. The response-time values can be sent unsolicited upon threshold overflow, or can be solicited by the network operator.

The network operator can request a summary of the response-time data for a specific LU over a user-defined period of time, detailed data for a specific session for a single collection period, and the long-term trend for a specific LU.

Managing logical resources

Because the protocols involve distributed participants, protocol violations and logical errors in SNA implementations can be very difficult to diagnose. Some information about the logical resources in an SNA network is captured by the control point and used for problem management.

Data captured by the control point on logical resources include session start information, normal session termination information, abnormal termination information (e.g., sense data), and virtual route information. This information is available for sessions that were started through the assistance of one or multiple control points.

With the advent of SNA network interconnection, it is possible to have a session established by a network wherein the endpoints of the session reside in different networks. In this case, additional session information is required for problem management. Retrieval of session information includes the gathering of session control-block information from gateway nodes (NCPs) for cross-network sessions.

Future Considerations

The requirements for SNA management services increase as SNA protocols and telecommunications complexity continue to advance; management services will need to be an integral part of each SNA enhancement. Common solutions are sought whether managing small or large systems interconnected via SDLC, X.25, or local-area networks.

A trend in both IBM SNA products and SNA management services has been to provide more granular monitoring and control of network components. Hence, such products as the 386x series of modems and the 3710 Network Controller offer extensive problem management features integral to their design. In turn, the architecture allows telecommunication links, for example, to be monitored and controlled at the level of their most basic subsystems: component adapters, modems, line concentrators, and transmission media. This has vastly improved problem management in SNA networks.

In the future, similar attention to management services features will be vital, particularly in the local-area networking arena. Management services enhancements for managing the logical resources would extend the ability to diagnose system-definition and implementation incompatibilities. The capability to trace routes as they are established dynamically, and to track users as they switch between sessions would also add to the serviceability of SNA networks.

LOCAL-AREA NETWORKS

Even as architectural solutions for improved connectivity continue at the higher layers of SNA, the announcement of the IBM Cabling System in 1984 promises major improvements in dynamic connections at the physical and data link control layers.

For the immediate future, the IBM Cabling System allows an SNA network to be physically reconfigured without running new coaxial cable to specific locations in a building. By using a structured-wire approach and wiring closets to prewire a building with either twisted-pair copper conductors or optical fibers, today's workstations can be moved from office to office by simply plugging them into a wall and reconfiguring at a conveniently located wiring closet.

Although ease of reconfiguration is a desirable goal, the ultimate objective is to eliminate entirely the need for manual intervention by a systems professional when moving a workstation from one office to another. This objective could be accomplished by the incorporation of a token-ring local-area network (LAN)[21] on the IBM Cabling System.

A token-ring LAN would consist of the wiring system, a set of communication adapters (*stations*), and an access protocol that controls the sharing of the physical medium by the stations attached to the LAN. The token-ring LAN is one of several LAN standards currently being developed by the IEEE 802 committee for submission to the International Standards Organization (ISO). (Other standards include one for

CSMA/CD on baseband cable and a token-bus standard on broadband cable.) A token-ring LAN is unique among these in that the nodes are physically connected serially by a transmission medium, such as twisted pairs or optical fiber. Access to the transmission medium is controlled through the use of a unique bit sequence (*token*) that is passed from one station to the next. When a station has a frame to transmit, it modifies the token to a frame by changing the bit pattern of the token to a start-of-frame sequence; the frame is then transmitted. When the station has completed frame transmission, and after appropriate checking for proper operation, it initiates a new token so that other stations have an opportunity to gain access to the ring.[22]

An important part of the token protocol is the ability of a station to reserve the token for use at a specified priority. This ensures that the next token issued will be at the highest priority requested, and allows a station to gain faster access to the ring for frame transmission than would otherwise have been possible.

To ensure that a token is always available on the ring, one station is elected as the *token monitor*. The function of the token monitor is to detect error conditions in token operation, such as a continuously circulating frame or the absence of a token on the ring. The capability to be a token monitor resides in each station, and is determined by an election process when normal token operation is disrupted.[23]

Several advantages exist in choosing a token-ring configuration for a LAN; these include the ease of fault isolation, performance stability under load, the use of predominantly digital rather than analog engineering, and the promise of optical fiber technology.[24]

To take full advantage of the peer-to-peer connection capabilities inherent in a shared physical medium, a station on the token ring could use the data link control, called a *logical link control* (LLC), as defined by the IEEE 802.2 committee for LANs. This LLC employs the Asynchronous Balanced Mode of operation (like that in HDLC) when a link connection is established, thereby allowing either station to send data link commands at any time, and to initiate responses independently of the other link station. This provides for a balanced type of data transfer between two link stations that operate as equals on a logical point-to point link;[25] the number of logical links sharing the same ring equals the number of distinct pairs of communicating stations.

When the ring reaches its capacity either physically, in terms of the number of stations it is capable of supporting for the required distance, or when the bandwidth is exhausted and the performance is not acceptable, a *bridge* can be added to combine two token rings into one logical ring. A bridge is a device that copies a frame from one ring and transmits it on the other. Bridges can be used to combine a number of small rings to preserve the integrated connectivity in an establishment while providing better fault independence and performance. Locating stations on a ring, or on multiple rings connected by bridges can be performed dynamically by broadcasting requests for specific station addresses. Once the station is located, routing data through bridges can be done efficiently by including the routing information to the destination station with each frame; this allows bridges to copy frames from one ring to another based on routing information in the frame format without building, referencing, and maintaining complex tables. Thus, expansion of the LAN to include additional rings need affect only the connecting bridge, and can be transparent to all other stations.

Other LANs could be connected to the token ring as well. Those that comply with the IEEE 802 standards could be connected using bridges similar to those used to connect two token rings. Thus, an SNA station implemented on a token bus could communicate to an SNA station on a token ring as though both were attached to the same LAN.

One desirable goal is the connection of LANs to the SNA backbone network. This could be done by several methods. For example, in an SNA network, any SNA node containing intermediate (forwarding) path control function could be attached directly to the LAN, thereby allowing SNA stations to gain access to the entire SNA network. Special gateways are not required because the LAN is providing the functions of the physical and data link control layers of SNA, and not higher layers such as path control. Connection of the intermediate-routing SNA node to the remainder of the backbone network could be accomplished either by channel attachment or via a remote communication link using SDLC.

The effect of the token-ring LAN on SNA would not be disruptive. To attach an SNA node to the token ring, very few changes would be required. Because SNA is a layered architecture, supporting a new physical and data link control layer has little effect on the rest of the architecture. Thus, path control, for example, could remain unchanged, thereby allowing SNA nodes attached to the token-ring LAN to participate in an SNA network immediately.

The long-term effects of LAN attachment could be far reaching. Improving dynamic connectivity and reducing system generation requirements in SNA products become even more important when physical connectivity in an establishment creates the possibility of a "hot pluggable," fully-meshed network. That is, once an SNA workstation is plugged into an office wall, it could have immediate physical access to all SNA workstations and other SNA nodes attached to the LAN. To translate this physical access into intelligent communication, a consistent application program interface and a set of protocols allowing peer attachment to work stations and mainframes, are critical. Thus, LU 6.2 and node type 2.1 protocols in SNA will become even more important.

LINK SUBSYSTEMS

SNA started with terrestrial links and channel attachments, and soon added dial capability. Local-area networking was discussed in the previous section. A number of other link-level options have been or could be accommodated by SNA. Some of these such as X.25, satellite technology, and ISDN, are described below.

Recommendation X.25 defines a packet-mode interface for attaching data terminal equipment (DTE) such as host computers, communications controllers, and terminals to packet-switched data networks (PSDNs). The International Telegraph and Telephone Consultative Committee (CCITT) introduced X.25 in 1976 and updated it in 1978, 1980, and 1984. IBM products that offer X.25 capability comply with the 1980 version of the interface. A PSDN provides connectivity to other DTEs using X.25 virtual circuits. Permanent virtual circuits provide fixed connectivity between DTEs, whereas switched virtual circuits provide dynamic connectivity using virtual call set-up and clearing capabilities. The X.25 interface also defines user facilities such as interface parameter negotiation, reverse charging, and closed user groups.

Having participated in the development of X.25, IBM announced the capability in 1977 for attachment of several DTE products to PSDNs in Canada and France. One of the early products was a network interface adapter, a stand-alone unit that converts between the link control protocol of SNA nodes (SDLC) and the X.25 protocols. This allowed most IBM products that communicate with System/370 hosts to use X.25. Initially, the communication controller (IBM 3705) for SNA System/370 hosts used a specific software adaptation for X.25. In 1980, an X.25 program product for the communication controller was introduced, allowing packet-switched communication with other SNA products and connections with non-SNA DTEs.

The IBM direction with respect to X.25[26] has been to integrate the interface into SNA products where required, so that the customer can choose the most economical communication medium. If network tariffs favor X.25, one can choose packet-switched services; otherwise, traditional leased or switched services can be used. By the end of 1984, 14 IBM products had been announced supporting the X.25 interface in more than 25 countries.

All SNA products that offer an X.25 interface conform to an IBM-defined specification[27] for attachment of SNA products to PSDNs. The most recent enhancement to this specification is Enhanced Logical Link Control (ELLC). Sometimes, virtual circuits are interrupted by the PSDN, causing inconvenience to the users of certain products. This inconvenience is reduced with the implementation of ELLC in low-end computers and terminals that provide a dynamic packet error detection and recovery procedure across one or more PSDNs between SNA nodes.

The IBM SNA X.25 interface specification will be updated to include aspects of the CCITT 1984 recommendation to be supported in SNA. Some of the new functions in the 1984 recommendation that are candidates for inclusion in the IBM SNA specification are

1. Multiple links between the DTE and the PSDN;
2. Forwarding a virtual call by the PSDN from the called DTE to an alternate DTE;
3. Hunt groups that allow the network to assign a call to one of several target DTEs; and
4. A subaddressing capability that allows a DTE on a private packet-switched network to call a DTE on a PSDN.

The SNA nodes discussed above are DTEs that attach to packet-switched networks. Some customers need equipment from different vendors to communicate with each other over packet-switched networks. If the customer has a mix of SNA and non-SNA traffic, the SNA backbone network could be used to carry the non-SNA traffic. Studies are underway to determine the requirement to add a PSDN appearance of the X.25 interface to SNA. In such a configuration, the SNA network would provide X.25 permanent virtual circuit, virtual call, and user facility services to using DTEs.

A key aspect of X.25 is that it is an interface specification, not a network architecture. The internal operation below the X.25 interface, such as routing, flow control, and management services, is not specified by standards. Inclusion of an X.25 DCE capability within SNA is a relatively straightforward and natural step.

Some customers find it advantageous to send their SNA traffic over satellite circuits. Because most communication satellites are in geostationary orbits above the equator at an altitude of about 23,000 miles, the delays in sending information from one earth station to another are long compared to those for a terrestrial circuit that connects the same two points on the earth's surface. Consequently, communication protocols must be designed to accommodate the long propagation delay of satellite circuits. Detailed studies of SNA[28,29] show that interactive and batch applications can use satellite links satisfactorily at speeds of up to 19,200 bits per second when the satellite link is attached to an SNA peripheral node.The IBM 3710 Network Controller provides satisfactory SNA performance over satellite links at speeds of up to 64,000 bits per second when the satellite link connects the 3710 to an IBM 3725 Communications Controller (using SDLC with a modulus of 128). Batch and interactive traffic over a satellite link that connects two SNA 3725 communications controllers is carried satisfactorily at link speeds of up to 256,000 bits per second. A special adaptation is available that allows two SNA 3725s to communicate over satellite links at speeds of up to 1.344 million bits per second.

At satellite speeds above 256,000 bits per second, special consideration must be given to the types of protocol and the value range of protocol parameters at several architectural layers of the system.[30,31] Because high-speed satellite circuits are becoming more widely available, a requirement exists to enhance SNA to accommodate them. Capabilities such as support for larger link-level sequence numbers (an SDLC modulus of 128) have been added; selective retransmission of information and other protocols optimized to the high-speed, long-delay environment are being studied.

The rapidly approaching feasibility of high-speed digital communication (in units of 64,000 bits per second) will have significant impact on both data communication and telephony, and will open possibilities for interactive video applications in the foreseeable future. The ISDN (Integrated Services Digital Network) standardization of the user-to-network interfaces for these applications has been going on for several years with CCITT.

IBM has consistently represented the needs of data communication applications in this effort. We continue to actively cooperate in the development of a single set of world-wide standards. The CCITT Recommendations of 1984 are a significant step in the advancement of these new digital transmission services.

SNA AND OSI

IBM recognizes the widespread interest on the part of users in interconnecting networks using different communication architectures, IBM favors such interconnection and publishes extensive information about SNA, including formats and protocols, which facilitates interconnection by other systems to IBM SNA networks.

Widespread interest also gave rise to the Open System Interconnection (OSI) standards project[32] aimed at providing communication protocols for interconnecting systems of different communication architectures, such as SNA with systems of other architectures. IBM has been involved from the start in this work. We have contributed what we have learned about layered communication architectures in the past several years, and increased our understanding of advances elsewhere in this area. Some capability for interconnecting heterogeneous systems is clearly desirable. From a vendor's perspective, a single international protocol for interconnecting heterogeneous systems is preferable to a number of national protocols.

IBM has stated that for industrial communications, IBM supports the National Bureau of Standards specifications for OSI Transport Layer Class 4 used over IEEE 802.4 LAN. The capability was demonstrated at the National Computer Conference in July, 1984.

IBM Europe has software under development that will provide IBM System/370 support for selected functions in the OSI 4 (Transport) and 5 (Session) layers. Testing in conjunction with third parties is planned to start during 1985. This represents a further step in IBM's commitment to provide IBM System/370 products capable of system interconnection in conformance with OSI standards.

OSI protocols could be implemented in SNA using a gateway concept; that is, we may transform the SNA protocols to and from the OSI protocols through gateway nodes in order to allow attachment to other networks. Indeed, IBM Japan, in cooperation with Nippon Telephone and Telegraph (NTT), has judged gateways using OSI as intermediate protocols to be a viable way to interconnect SNA networks with networks that use the DCNA protocols.[33–35] This approach of using gateways, as opposed to adopting OSI protocols as internal operation protocols for SNA networks, reflects the fact that OSI protocols today are intersystem protocols, as opposed to network architecture.

CONCLUSIONS

SNA, now eleven years old, has evolved continually and will do so as long as new technology, applications, and requirements unfold. The layered structure of the architecture and of the implementing products allows this process to be natural and nondisruptive.

We have cited some of the historical trends. For example, configurational flexibility and accommodation of larger networks have been ongoing concerns. In addition to the generalized topology and larger address space now available, specific offerings such as the recent IBM 3710 Network Controller, which offers a new remote link concentration capability, have resulted in cost-performance advantages.

Additional offerings have resulted in the extension of SNA capabilities into areas that are important to many customers. A recent example is the inclusion of VTAM as an integral component of the native VM environment, thereby enhancing performance of SNA network operation from the VM viewpoint. Another area of traditional concern to customers is non-SNA device support. One technique, which uses format envelopment, was incorporated into the Non-SNA Interconnection (NSI) program product on the NCP; it allows BSC remote job entry terminals and BSC network job entry subsystems to communicate through an SNA network and share the SNA links. Another technique employs protocol conversion; aside from the long-time NTO support for pre-SNA terminals in the NCP, new capabilities such as those of the 3710 allow concentration of start-stop links onto SDLC links, and continue to enhance SNA coverage in this important area. The trend in general has been to perform protocol conversion to SNA as close to the non-SNA interface as possible in order to gain the SNA benefits of resource-sharing and network management quickly in the operation.

Of course, one of the most visible areas of non-SNA protocol support is national and international standards. IBM will continue to cooperate in the formulation of such standards, and include support for such standards in SNA products subject to appropriate business decisions.

Network management services have generally been and clearly must continue to be integrated into SNA and its implementing products. This commitment increases as the richness of the architecture grows; to control problems associated with increasing complexity, the pace of extensions in this vital area will likely be stepped up.

Other trends will become more prominent. The need for continuous network operation will foster ever more features that promote high availability. The Extended Recovery Facility mentioned earlier in this paper is an archetypal feature in this category. Advances in route dynamics and distributed directories will also play a significant role in meeting requirements in this area. The matter of reducing the static nature of network definition is an ongoing concern; a long-term goal is to eliminate the need for static definition entirely.

Finally, SNA will continue to exploit advances in technology as they come along. A known requirement is to extend the peer-to-peer operation in SNA. This need follows from developments both in processor design, especially in small systems, and in different types of transmission technology, such as local-area networks, satellites, PSDNs, and ISDN. Other developments, not yet evident, will affect future requirements. The process will continue and will undoubtedly cause much interesting evolution of SNA for a long time to come.

REFERENCES

Note: All IBM publications, unless otherwise noted, are available through IBM branch offices.

1. Sundstrom, R. J., and G. D. Schultz. "SNA's First Six Years: 1974–1980." *Fifth International Conference on Computer Communication.* Amsterdam: North-Holland, 1980, pp. 578–585.
2. Amdahl, G. M., G. A. Blaauw, and F. P. Brooks. "Architecture of the IBM System/360." *IBM Journal of Research and Development,* 8-2 (1964), pp. 87–101.
3. Dijkstra, E. W. "The Structure of the THE Multiprogramming System." *Communications of the ACM,* 11-5 (1968), pp. 341–346.
4. Benjamin, J. H., M. L. Hess, R. A. Weingarten, and W. R. Wheeler. "Interconnecting SNA Networks." *IBM Systems Journal,* 22-4 (1983), pp. 344–366.
5. IBM Corporation. *SNA Format and Protocol Reference Manual: SNA Network Interconnection.* SC30-3339, 1985.
6. Eisenbies, J. L., and T. D. Smetanka. "An Automatic Topology Update Scheme for SNA Networks." *Proceedings of the Seventh ICCC.* Amsterdam: North-Holland, 1984, pp. 725–730.
7. Jaffe, J. M., F. H. Moss, and R. A. Weingarten. "SNA Routing: Past, Present, and Possible Future." *IBM Systems Journal,* 22-4 (1983), pp. 417–434.
8. Baratz, A. E., J. P. Gray, P. E. Green, J. M. Jaffe, and D. P. Pozefsky. "SNA Networks of Small Systems." *IEEE Journal on Selected Areas in Communications,* SAC-3, 3 (1985).
9. IBM Corporation. *SNA Transaction Programmer's Reference Manual for LU Type 6.2.* GC30-3084, 1983.
10. IBM Corporation. *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2.* SC30-3269, 1984.
11. Gray, J. P., P. J. Hansen, P. Homan, M. A. Lerner, and M. Pozefsky. "Advanced Program-to-Program Communication in SNA." *IBM Systems Journal,* 22-4 (1983), pp. 298–318.
12. Sundstrom, R. J. "Program-to-Program Communications—A Growing Trend." *Data Communications,* 13-2 (1984), pp. 87–92.
13. Housel, B. C., and C. J. Scopinich. "SNA Distribution Services." *IBM Systems Journal,* 22-4 (1983), pp. 319–343.
14. IBM Corporation. *Systems Network Architecture Format and Protocol Reference Manual: Distribution Services.* SC30-3098, 1984.
15. Schick, T., and R. F. Brockish. "The Document Interchange Architecture: A Member of a Family of Architectures in the SNA Environment." *IBM Systems Journal,* 21-2 (1982), pp. 220–244.
16. IBM Corporation. *Document Interchange Architecture: Concepts and Structures.* SC23-0759, 1983.
17. Cordell, R. "Rock-Solid Office Architecture." *Computerworld on Communications,* October 3, 1984, pp. 21–23.
18. IBM Corporation. *Document Content Architecture: Revisable-Form-Text Reference.* SC23-0758, 1983.
19. IBM Corporation. *Document Content Architecture: Final-Form-Text Reference.* SC23-0757, 1983.
20. IBM Corporation. *SNA Format and Protocol Reference Manual: Management Services* (forthcoming publication).
21. IEEE. *Token-Ring Access Method and Physical Layer Specifications.* Project 802, Local-Area Network Standards, Draft IEEE Standard 802, Draft E, August 1, 1984.
22. Andrews, D. W., and G. D. Schultz. "A Token-Ring Architecture for Local-Area Networks—An Update." *Proceedings of COMCON Fall '82.* Los Angeles: IEEE Computer Society, 1982, pp. 615–624.
23. Strole, N. C. "A Local Communications Network Based on Interconnected Token-Access Rings: A Tutorial." *IBM Journal of Research and Development,* 27-5 (1983), pp. 481–496.
24. Saltzer, J. H., K. T. Pogran, and D. D. Clark. "Why a Ring?" *Computer Networks,* 7 (1983), pp. 223–231.
25. Carlson, D. E. "Bit-Oriented Data Link Control." In P. E. Green, Jr., (Ed.), *Computer Network Architectures and Protocols.* New York: Plenum Press, 1982, pp. 111–143.
26. Deaton, G. A., and R. O. Hippert. "X.25 and Related Recommendations in IBM Products." *IBM Systems Journal,* 22-1/2 (1983), pp. 11–29.
27. IBM Corporation. *The X.25 Interface for Attaching IBM SNA Nodes to Packet-Switched Data Networks—General Information Manual.* GA27-3345, 1985.
28. Deaton, G. A., and D. J. Franse. "Performance Analysis of Computer Networks that Access Satellite Links." *ICCC-80 Conference Proceedings.* Amsterdam: North-Holland, 1980, pp. 297–302.
29. Deaton, G. A., and D. J. Franse. "Analyzing IBM's 3270 Performance over Satellite Links." *Data Communications,* 9-10 (1980), pp. 117–132.
30. Andrews, D. W. "Throughput Efficiency of Logical Links over Satellite Channels." *Proceedings of the Satellite and Computer-Communications Symposium.* Amsterdam: North-Holland, 1983, pp. 231–241.
31. Brodd, W. D., and R. A. Donnan. "Data Link Control Requirements for Satellite Transmission." *Proceedings of the Satellite and Computer-Communications Symposium.* Amsterdam: North-Holland, 1983, pp. 201–213.
32. Day, J. D., and H. Zimmerman. "The OSI Reference Model." *Proceedings of the IEEE,* 71-12 (1983), pp. 1334–1340.
33. Shukuya, S., M. Yamaguchi, and Y. Kobayashi. "A Study of OSI Subsets from LU 6.2 Functional Viewpoints." *Proceedings 30th Annual Convention IPS Japan.* Tokyo: Information Processing Society of Japan, 1985, pp. 1053–1054. (In Japanese.)
34. Yamaguchi, M., and K. K. Sy. "A Comparison of Various Gateways for SNA-OSI Interconnection." *Proceedings 30th Annual Convention IPS Japan.* Tokyo: Information Processing Society of Japan, 1985, pp. 1055–1056. (In Japanese.)
35. Shiohara, M., K. K. Sy, and M. Yamaguchi. "A Study of the Protocol Conversion between SNA LU 6.2 and OSI Session/Transport Layers." *Proceedings 30th Annual Convention IPS Japan.* Tokyo: Information Processing Society of Japan, 1985, pp. 1057–1058. (In Japanese.)

Token passing local area networks: A success story for standards

by MARY JANE STROHL
Concord Data Systems
Waltham, Massachusetts

ABSTRACT

This paper presents the software architecture and the software development approach used by Concord Data Systems (CDS) for the development of their IEEE 802.4 standard token passing local area network products. The paper also describes the design goals and features of the CDS implementation of the ISO transport protocol. The transport implementation features a flexible interface that supports, transparently, both on-board and off-board session entities. Finally, the CDS services for network management and Layers 5–7 of the ISO/OSI Reference Model are discussed to show the current features and a migration approach for including standard protocols for these layers as they mature and become accepted standards.

INTRODUCTION

Beginning with a design commitment to develop token passing local area network (LAN) products that are compliant with ANSI and ISO standards, Concord Data Systems (CDS) developed a modular software architecture based on the International Standards Organization's Open System Interconnect (ISO/OSI) Reference Model. The architecture supports an incremental approach to building and integrating standard protocols into products, and has worked successfully to produce a versatile set of high-performance LAN communication products.

Token passing network applications vary from factory control applications to interactive computer terminal servers. This diversity in applications dictates communication architecture alternatives for distributing the OSI upper layer software across hardware boundaries. These alternatives provide both LAN network interface units that serve as front end communication processors for an attached host, and LAN units that function as end systems for applications such as the terminal server.

This paper presents the software architecture approach that CDS has used for developing and integrating standard protocols into its LAN products. It is a building block approach with an interface design concept that gives the product set its versatility. The development of the OSI Layer 4 transport protocol, shown in Figure 1, is used as an example to show how the architecture supports independent development testing integration and performance analysis for one of the OSI Layer Protocols.

The role of Layer 5–7 protocols and network management software in token passing LAN products, shown in Figure 2, is discussed next. Standards are just beginning to evolve for these layers. The status of these standards for LANs and an evolutionary development strategy for them are presented.

A BASIC ARCHITECTURE FOR LOCAL AREA NETWORKING SYSTEMS

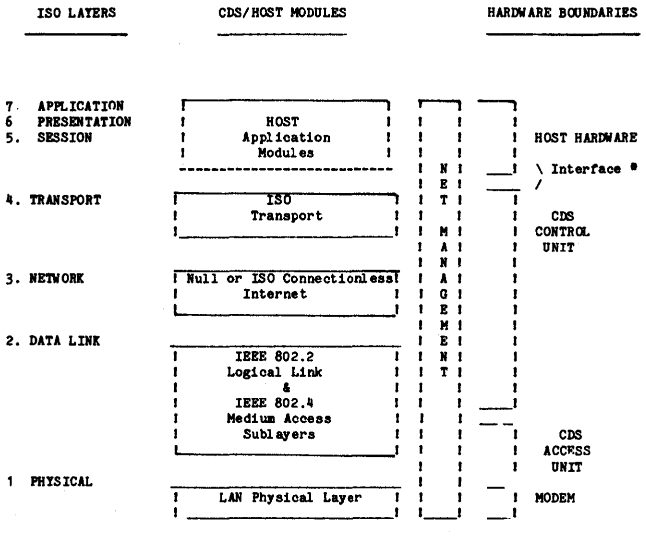
The CDS architecture approach was required to offer standard IEEE 802.4 high-performance networking products that meet a variety of application requirements. To achieve this, the CDS design focused on interfaces, a modular architecture based on the OSI Reference Model, the use of standard protocols, and performance.

Interface considerations are important to satisfy a wide variety of application requirements. LAN interface units can encompass a range of products, from simple access units where Layers 3–7 are provided by a host computer, to stand alone devices where all seven layers of the ISO/OSI model reside in the LAN unit. An attractive division of functions is one where the LAN unit is truly a communications processor with the communication layers 1–4 in the LAN unit and the application related layers 5–7 in the host (Figure 1). Off-board interfaces to the network layer or possibly network sublayers are other interface possibilities. There are applications to support all of these physical architectures, but to do this effectively in a product line requires well defined and flexible interfaces. The interfaces must be such that the transport layer can service both on-board and off-board session entities.

With a modular design where interfaces are designed correctly for the first release, upgrades to the modules themselves can be made without making interface changes, giving *plug-compatible* software module versions. Then, with a download capability, product maintenance becomes reasonable. Products can be maintained by downloading new versions of the upgraded modules. This approach was taken with the design of the LLC code in the CDS LAN unit, and worked successfully. The code supports an HDLI (the General Motor's Manufacturing Automation Protocol (MAP) data link protocol that has a flow control option along with the standard HDLC LAP B protocol) interface to provide host access to the LLC module as well as the on-board network and transport interfaces to LLC.

Along with well-designed interfaces, modularity is another critical design requirement necessary to make this architecture work. The OSI Reference Model, with its layering concept, forces a basic modular architecture. Within the layers, the code design must also be modular. To help realize this modularity in the CDS software design, Pascal was the chosen implementation language. Because of its data typing features, Pascal constructs can be used in a design specification and fold directly into an implementation. The First Systems Pascal (by First Systems Corp., Manhattan Beach, CA) that was used by CDS offers ADA-like concepts, such as an equivalent to the ADA Package, and facilitates large project developments. The philosophy was to develop a modular implementation in Pascal that would give a tight, maintainable, and self-documented code architecture. This could then be used to selectively do assembly recodings of small modules where performance was critical.

The incorporation of standards in the CDS architecture closely followed the developments in the standards communities. With LANs, the first standards to evolve were for Layers 1 and 2, and were driven by the IEEE 802 committees. The medium access (MAC) sublayer and physical specifications for CSMA/CD, Token Passing Bus, and Token Ring networks were specified by IEEE 802.3, 802.4, and 802.5, respectively. The logical link (LLC) sublayer of the OSI data link layer was



* Interface is a shared memory or external interface.

Figure 1—Host/CDS transport interface diagram

specified by IEEE 802.2. These early standards for the lower layers of the ISO/OSI Model allowed LAN developers to build their network interface units on a standard Layer 1 and 2 base.

The Layer 4 Transport service matured at about the same time as the lower layer standards. While the military and the ARPANET community have adopted TCP as their transport protocol (with IP as its internetworking companion for Layer 3 services), the rest of the LAN world seems to be a mixture of TCP, OSI Transport, and XNS (the Xerox Protocols). The acceptance of the OSI protocols by General Motors and NBS in the MAP development and the bandwagon they have gathered as a result of their sponsorship of the Multivendor Demonstration at NCC '84 have focused vendor development at least in manufacturing applications on the OSI protocols.

These developments in the standards community allowed the CDS architecture to evolve from an initial prototype product with standards at only Layers 1 and 2, to the product shown in Figure 2. The current product supports the IEEE standard protocols at Layers 1 and 2, a null Layer 3, the ISO transport at Layer 4 and a Serial Port Application at the higher layers. An awareness of and strong participation in the standards work was beneficial to the design phase of the product. ISO Internet and the Connectionless ISO transport, for example, were anticipated, and the design allowed for the eventual support of these protocols. The same is true for the standards work in Network Management where the company has been active in the work of the IEEE 802.1 committee.

The LAN interface units as communications processors or end systems must offer both high performance and low cost. The cost requirement dictates microprocessor architectures. The high performance requirement dictates tight designs that are performance driven. In some cases, the overhead of the multilayers of the OSI model and its large protocols (transport, for example) add an extra and often controversial burden to the performance (throughput) goals. With a good basic modular design, performance enhancements can be gained from the distribution of selected functions to hardware, firmware, or assembler modules. There is also enough richness in the different classes of transport and LLC services to determine, through performance analysis, an optimal configuration given the underlying network's reliability and its application requirements.

To realize the high speeds offered by the LAN access units, the higher layer communication software must also offer high performance. An efficient real-time executive (exec) is an essential ingredient for providing this performance. The CDS exec is a small multi-tasking exec based on MTOS, the Multi-Tasking Operating System. The exec has a simple task scheduling algorithm and service primitives that allow tasks to disable the scheduling mechanism during critical sections of code.

THE CDS TRANSPORT SERVICE

The CDS Transport Features

The primary design goals for the CDS transport implementation, like those for other CDS modules, were modularity, ease-of-use, maintainable interfaces, testability, and performance. Adhering to the OSI Transport Standard, and providing configuration options and counters for network management were critical functional requirements of the design.

The CDS transport implementation is a conformant superset of the ISO DIS 8073 Transport Specification. It offers both the ISO standard connection-oriented service and a connectionless datagram service. The connection-oriented transport classes 2, 3, and 4 are offered in the CDS transport implementation and are fully compatible with the ISO specification. In addition, this transport implementation offers compatibility with the NBS based but ISO compatible MAP transport specification.

The connection oriented transport services provide transparent and reliable data transfer between two transport users. Transport provides a negotiated quality of service by matching the requirements of the user application to the characteristics of the underlying network and providing enhancements, where required, to the network service. The enhancement levels are associated with the transport classes. Five classes, 0 through 4, provide minimal (Class 0) enhancement to increasing levels of error detection and recovery with increasing Class number.

The CDS connection oriented transport implementation supports connection negotiation and expedited data for transport classes 3 and 4, as required by the standard. Connection negotiation allows the transport users to negotiate characteristics such as the transport class, the maximum transport frame size and the use or non use of check sums for their connection. Expedited data allows short messages, up to 16 bytes, to bypass the normal flow control mechanisms of transport and to be delivered ahead of any other non-expedited messages in the transport queues.

The connectionless transport datagram service provides an efficient, low overhead data transport service. The datagram service is not a reliable one because messages are subject to loss and duplication and are not flow controlled at the transmitter. The underlying 802.4 data link services are reliable, however, so that the service is a reliable one from the user's point of view. It is particularly useful for group broadcast services like those required by a name service. The CDS datagram service is based on the current work in the standards community on the connectionless transport service. The implementation will migrate, where necessary, to full compatibility with the ISO connectionless transport service when it reaches a standard status.

The CDS Transport Interface

The interface between the LAN transport layer and the application transport user is called the *Communication-Application Interface.* It is designed to provide a communications path between cooperating applications at one or more locations. The key design feature of the interface is its ability to commonly support both applications that reside on the same board as transport and applications that reside offboard. This gives a shared memory transport interface to support host-based applications. The transport service interface can also be ported to a host and carried via an external interface to the LAN unit.

The Communication-Application interface is a shared-data interface whose primary data structures are communications control blocks (CCBs) and request control blocks (RCBs). These structures are linked together on queues that are connected by a CCB base which, together with an event flag mechanism, provide the interface backbone. A set of service routines for control block initialization queue handling and flow control between transport and the application are the tools required to manage this interface.

The design has a flexible buffer allocation strategy that allows the transport user to allocate its buffers to transport according to the session requirements. At the time the CCB is initialized, the transport user allocates buffers to the CCB's free-queue and indicates how these should be used by transport by issuing credits for its *receive* queues. There are separate queues associated with each CCB for expedited data items and for normal data and control items, and for transport to return issued requests with their status.

Flow control is based on receiver allocation of buffers. The transport layer provides a credit based service that allows transmission over a virtual circuit only if buffers are available at the receiver. A quota field that is shared between transport and the transport user (in ISO, the session layer) is part of the CCB. It provides the credits to the communication layers for received data blocks and is used by transport for credit computations on flow controlled connections. The quota is decremented by transport each time a block is removed from the CCB's *free-data* queues for a received data frame. The application initializes this field and increments this quota each time a data block is returned to the free-data queue.

Transport Performance Considerations

Extensive performance analysis, down to the module level, was done on the CDS transport implementation. The effort identified unreasonable module activity and demonstrated the task dynamics. Assembler recodings of selected modules and adjustments of relative task priorities gave significant performance enhancements. Optimizing transport's use of the exec services also helped to raise the level of the transport performance.

The question of the appropriate transport class and the set of options within a class for LANs, particularly single-hop LANs with no internetworking, has been addressed from a performance point of view. LANs generally offer a communications media characterized by high speeds and low error rates, so the extensive error detection and recovery features of Class 4 Transport may be excessive for some applications. Transport check sums, for example, are costly in terms of performance, and we have demonstrated that they are not necessary to provide reliable end-to-end transport connections on a single-hop LAN. CDS is currently evaluating the performance characteristics of transport with the various classes and options.

Layers 5–7 and Network Management in LANs

The CDS port application, called *SPA,* offers session and presentation services to a terminal user as seen in Figure 2. It interfaces directly to the transport layer and was designed so that the session services in particular could be replaced by a standard session protocol. The SPA session service supports a variety of session types, including fixed or switched circuits, and point-to-point or multipoint connections. Both asynchronous and transparent synchronous data transfers are supported. SPA provides session negotiation and has a rotary

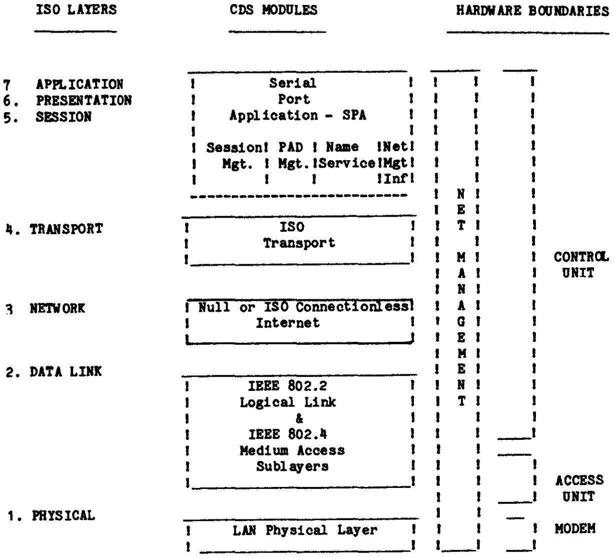


Figure 2—CDS "Terminal server" architecture

function that selects the *best* port for a given call request. In addition to the session service, SPA offers a name service that allows remote ports to be addresses by configurable names. The name service uses the datagram service of the transport protocol. A station management component supports the network management entities' access to internal port application configuration parameters and counters. As an application, SPA can be considered to be a packet assembler/disassembler (PAD) with features similar to those of an X.3 PAD.

The network management facility of the CDS architecture features both local and remote reading and configuration of the parameters of the network interface units. A CDS network control center (NCC), an IBM XT based system, executes the remote procedures with cooperation from the remote station management entities for each software layer, and maintains the network management database with a relational database system. The network management architecture was designed along the guidelines of the IEEE 802.1 work, and will eventually use the management protocols that are being developed in 802.1. Both local load of software and the 802.1 remote load will be supported. The interface, like the transport interface, will be a shared data queued structure for the management transactions.

Activity in the standards community is now focusing on layers 5–7 and network management. It is likely that standards for these layers will emerge and become accepted within the next few years. The need for application support drove early LAN implementations so that an evolutionary plan is required to have LAN products that incorporate standards for the upper layers as they become accepted. Again, a product architecture based on the OSI Model proved to be a wise choice, as early products with non-standard layer implementations can be replaced by standard protocols—thus ultimately achieving a standard product within the original product architecture.

SUMMARY AND CONCLUSION

By adopting a software architecture based on the ISO/OSI Reference Model, an incremental and modular approach to the development of LAN products can be taken. The payoff is products whose base components meet accepted standards, offer interoperability with other vendors' standard products, and provide an evolutionary path for the development of completely standard products as the standards emerge for the higher layers. The architecture then becomes a frame into which the building blocks can be inserted as they are developed.

CDS has used this approach successfully to develop the software architecture for its product base. The ISO transport protocol implementation is one of the standard building blocks that has been incorporated into the original product that was based on the standard IEEE 802.4 token bus access protocols. The approach also provides an evolutionary path for the incorporation of the standard network session and network management protocols.

REFERENCES

1. *Information Processing Systems, Open Systems Interconnection,* Basic Reference Model, Ref. No. ISO 7498-1984 (E). New York: American National Standards Institute, 1984.
2. (Draft) IEEE Standards 802.2, 802.3, 802.4, 802.5. Silver Spring, Md.: IEEE Computer Society, 1985.
3. *Draft IEEE Standard 802.1, Part A—Overview and Architecture. (Revision B, June 1983). Part B—Systems Management (Revision G, January 1985). Sponsor.* IEEE Computer Society.

Protocol implementation strategies in local area network access units

by WENDELL TURNER
TRW Information Networks Division
Torrance, California

ABSTRACT

Local area networks are implemented with microprocessor-based access units that not only establish and maintain virtual connections over a shared medium but also provide the person-machine interface to the network and its services. The protocols implemented are built to take advantage of the topology and underlying data transfer mechanisms. The transport layer is not needed during data transfer over a virtual connection because of the point-to-point nature of the network and its inherent differences from store and forward networks. Net management may be accomplished either by session-level datagrams or by a virtual connection to an application layer task.

INTRODUCTION

This paper will discuss higher-layer protocols and their implementation in local area networks. These networks are different in many respects from other networks, and the protocols designed can take advantage of this difference to make their implementation easier and their operation efficient. First discussed will be some of the differences in these networks and the advantages that can be realized from these differences. The higher-layer protocols of the ISO Reference Model[1] will be examined along with some possible implementations.

SOME BASIC PROPERTIES OF LOCAL AREA NETWORKS

Local area networks, or LANs, are confined to a small geographic region, usually less than 10 miles in diameter. This allows a topology that directly covers the region; the most common topologies are the bus and the ring. With either one of these topologies every network access unit has a direct connection with every other unit. The point-to-point nature of the network allows the protocols certain liberties with the way they handle messages; this topic will be discussed later. There are currently no store-and-forward LANs, and, although one may be developed, these LANs will not be addressed here. Some local networks do contain repeaters, but these repeaters have storage for no more than a few bits and perform no routing function.

Local area networks are owned and operated by a single organization. They are not public networks and are not regulated by any government agency, and thus they do not necessarily have to be open systems allowing interconnection at all levels. Some of the requirements put on the protocols can be relaxed so that they can be more suited to the applications.

The network will be used for office automation, data processing, and manufacturing control. Some LANs also transmit packetized voice and video. There will be significant bursty interactive traffic and some file transfer or host-to-host activity. The most common access to the network will be through a network access unit (NAU). The NAU connects a computing device or a peripheral to the network and implements all protocols used on it. The NAU is a microprocessor-based control unit that can perform input-output operations to a terminal port (via RS-232, current loop, or other type of terminal protocol) and the network, performing either carrier sense multiple access (CSMA), token bus, or token ring access to it. A terminal can be connected to the NAU and, with the appropriate commands, can set up a virtual connection to any other port on the network.

The easiest way to connect the network to the host computer is through a NAU connected to the host as if it were a local terminal. The implementation of this connection often amounts to a rack of NAUs connected to the host's multiplexor through a bundle of RS-232 cables. Because of the physical appearance of such connections, this is called the "milking machine" approach to computer networking.[2] It allows remote access and remote login to the host computer, but other possibilities are limited by the host's software. It is sometimes difficult if not impossible in this arrangement to have the host initiate the transferring of a file or the sending of mail, so the host is limited in what it can do with the network.

Another method of host connection is a front-end processor that understands both the network protocol and the host computer's internal channel or bus. This allows for a much faster interface and does away with the need for a multiplexor. However, to do this the NAU would have to be much faster and larger than a terminal NAU and would have to be integrated intimately with the host software and hardware. This may not be feasible in many applications, so the "milking machine" approach is often used.

Network access units are sold in a highly competitive marketplace. They are designed to be a low-cost device, and therefore they do not have an excess of processing capability or memory. The services and functions that each protocol layer provides must be integrated with the other layers, resulting in a cohesive and minimally redundant set of services that is efficient and cost effective to implement.

The following sections discuss the transport, session, presentation, and application layers and their implementation in a typical LAN. Discussion here will be restricted to the networks as defined; the special properties of bridges, gateways, and internetting will not be addressed.

Transport Layer

Consider first the functions of the transport layer. The transport layer protocol is responsible for naming and addressing, connection establishment and termination, flow control, buffering, multiplexing, and error recovery.[3] Some local network protocols make use of the transport's datagram services for data transmissions that require the reliability of a virtual connection.[4] The transport layer described in Reference 4 provides a datagram service that depends on some of the unique properties of local network topologies. Discussed here is an alternative implementation strategy that shows that the lower protocol layers already provide the reliability needed for virtual connection and datagram traffic; the transport layer is responsible for managing the connections.

The transport protocol must provide a set of transport addresses or sockets for the higher layers to use. These socket identifiers form a networkwide unique name space. Other protocol processes can request that one transport socket be connected to another transport socket, forming a transport connection between the two. This service is provided by connect, listen, close, send, and receive primitives. The connect, listen, and close primitives are used for connection maintenance, the send and receive primitives to send data over the connection. As will be described, the transport layer's virtual connection functions are duplicated at other layers and the transport task can be passed by for traffic over the connection, saving several context switches and data copying operations for each message that traverses the internal data path. The transport layer's services are needed only to establish the call or to terminate it.

Positive acknowledgment and retransmission of data frames are provided by the transport layer if the subnet is unreliable. Because of the topology of these networks there is no storing and forwarding of packets, and thus NAU-to-NAU acknowledgments are end-to-end acknowledgments. (An acknowledgment in a CSMA/CD system, for instance, is the collision-free transmission of a packet.) In addition, there can be no loss or duplication of packets between transport programs, because the programs reside in the same "host" as the lower-layer software. With guaranteed delivery by the lower-layer software, the transport layer is relieved of this responsibility.

Emerging standards for link-level protocols employ function codes that indicate the buffer availability of the receiver.[5] The function codes provide flow control from one NAU to another. Since there is no buffering in store-and-forward nodes, the only buffering takes place in the source or destination access units. Assuming that buffer control internal to a NAU is not a problem and can be handled easily, end-to-end flow control is not a problem; therefore the lower layer's flow control will suffice.

The Reference Model allows the transport layer to multiplex data from several internal locations into the same network packet for transmission. This conserves bandwidth and channel access times, but due to the nature of local networks, bandwidth is relatively cheap and does not need to be conserved in such a manner. Also, multiplexing data in packets requires some copying, which is not efficient for every packet on such a small processor. The processing speed and buffer memory may not allow layers to copy data to new buffers; therefore each layer can add its header. In this way the data format will be integrated vertically throughout the access unit; it is not completely independent for each layer.

The transport layer must also provide error recovery for data. A cyclic redundancy check, or other checksum—one means of detecting damaged packets—is computed for each packet of data. Since multiplexing has already been discarded, the checksum is computed on a single user's data. But the data link layer computes checksums on all packets; and, given the point-to-point nature of these networks, the checksum computed by the transport layer is not needed as a check on network transmission validity.

Given the unique nature of LANs, the transport layer can be optimized as has been shown. With either the bus or ring topologies, the network access units can access others on a point-to-point basis, eliminating many of the transport layer functions in providing a reliable end-to-end service. With reliable data transfer now established, the rest of this discussion will concentrate on the higher layers of protocols and their roles in connection establishment and network management.

Session Layer

The main purpose of the session layer is to add user-oriented services to the transport layer. Using the facilities provided by the lower layers, this layer adds to them functions that will make network use and maintenance easier. In addition to maintaining virtual connections, it also provides translation of names known by the source application program into networkwide unique addresses of processes or ports; and it provides useful and timely network management features, including status monitoring and software reconfiguration. Some of the session layer functions will not directly involve protocols sent over the network but will be useful in the management of network operation.

The most common function of the session layer is to establish and terminate virtual connections. It must accept a connect request from the application layer programs, reformat it if necessary, query the resources on the net, acquire the desired resource, and establish the internal connection between the lower-layer software protocol entities and the application program that requested the connection. From this point on, the session layer does not participate in moving traffic over the virtual connection. When the application program has finished using the connection, it will signal the session task that the connection should be removed. The session task then signals the peer process in the other NAU that the connection is to be removed. Both session tasks perform the necessary processing to remove the connection and internally restore data paths and process states to their original condition. Figure 1 shows an example of the data connections established by the session layer during a virtual connection.

Obviously, the session layer must be receptive not only to requests for network activity from application programs residing in the same NAU, but also to unsolicited incoming requests for connections and for other session services. There is a routine in the session layer, called a session service agent, that will constantly listen for unsolicited requests from other session layers. In the particular case of a connection request, the incoming connection request, if honored, causes the resource identified to be marked as busy if it is a physical resource (I/O port), and causes a process activation if it is to a software resource. In either case the proper response is formatted by the session service agent and returned to the caller, and internal connections are established that route data received over the network to the identified entity.

Integrated into the call placing and answering is a logical name-to-physical-address translation. It would be useless to try to keep an internal list of the logical name and physical addresses of all the connectable entities on the network, because it would consume too much internal memory and need constant updating. A dynamic approach is taken to resolve this problem.

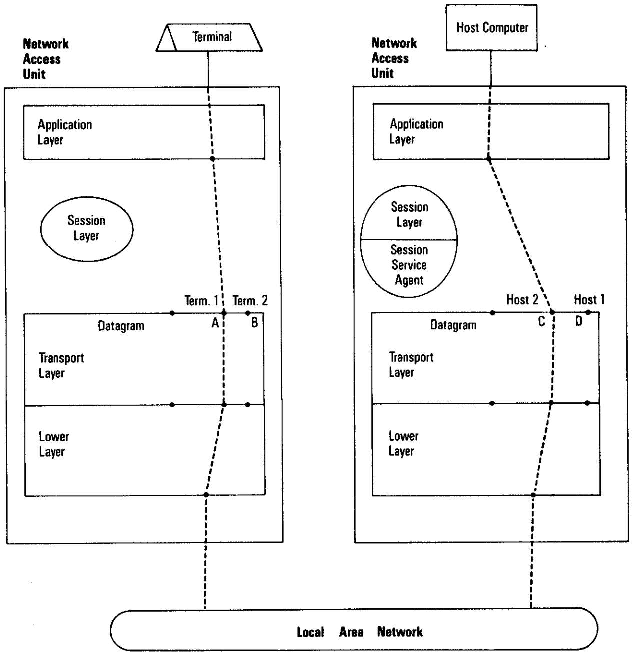


Figure 1—Data flow over a virtual connection

An internal query operation using broadcast datagrams will return the status and network-unique name of a resource identified by a given character string. The query operation performed by the destination session service agent in response to a query operation is built with a "rotary" feature, similar to a conventional office telephone. In trying to establish a connection to one of the 16 ports on a host computer's multiplexor, for example, any one of the 16 will do, but each has a unique address. The rotary feature gets around this by allowing a connection to the first available port. In this way similar resources can be logically grouped together.

The response to the name query will be the full name of the resource, its status, and its physical network address. The session layer sending the query then receives the responses, filters out the one needed, and processes it for the connection. Flooding is one of several techniques used successfully in implementing this sort of dynamic approach to networkwide query operations. After the entity to connect with is identified, the connection request is sent to the selected session service agent.

During this process the session service agent must check many items before allowing the connection. Certainly, if the requested port is busy, then the request must be denied; but passwords and privilege levels may also cause denial. Internal memory and processing capabilities may allow a maximum number of software resources to be running simultaneously; the first few requests will be honored, and the others will be denied.

The session services relating to virtual connections and their maintenance have been discussed. Network management and the functions and operations involved will now be discussed.

Each network access unit has many associated parameters. Terminal port parameters such as baud rate and stop bits, as well as port names, must be configured for each unit. Each layer of the protocol hierarchy must be properly configured, and each has statistics of its activity to report. There are many ways to read and set these parameters on the access units; two methods will be described here. The first, which uses the session service agent, is called *remote operations*. The second, which makes use of the connection abilities of the application layer, is called *remote command processing*. Remote command processing will be described in the section on the application layer.

In the remote operations scheme, the command entered is parsed by the local application command processor. A query or datagram request is then built, depending on the particular command, and sent to another session service agent. It is referred to as remote operations because the packet containing the command is sent to a remote NAU to perform the operation.

ISO Draft Recommendation X.409[6] has formally defined a structure for the internal representation of tables needed by the various protocol entities in a NAU. This format, if used, rigorously structures all tables inside the NAU. With all tables in an identical format it is relatively easy to define operations to access and manipulate those tables.

ISO Draft Recommendation X.410[7] defines a structure for defining the operations on those tables. It includes the operation definition, its parameters, and its status return code definitions. The X.410 protocol works in conjunction with the X.409 structure definition to manipulate elements in any layer in any NAU.

To assist in the overall view of the effect of this kind of structuring, the parameters in the tables belonging to an access unit can be conceptualized as tuples in a relational database. The session task then provides query retrieval, updating, and partitioning operations on the network. The session task residing in the same NAU as the requestor will format the request into a transport datagram and send it to another session service agent.

The session service agent receives the query and examines its own internal data as if those data were a single tuple in a database. The service agent formats the responses according to the X.410 Recommendation and reports them to the requestor. Note that this is a distributed processing event in that every tuple of the database has its own processor. The local session service can request information either explicitly or generally. An explicit request identifies a particular NAU and solicits information from it. A general information request is sent via a broadcast datagram or group address datagram, and each session service agent receives the request and responds to it only if the request can be satisfied. If an explicit request is made that spans more than one NAU, the local session service breaks down the request into an explicit request for each affected session service agent. This makes it easy to, say, set all ports that have the logical name "host 1" to a baud rate of 9600, or display the names of all ports to which a connection can be established. Figure 2 shows the data flow during a typical operation of this kind.

Through logical naming and password protection the network can be logically partitioned into groups, allowing some users limited access to all the resources on the network and

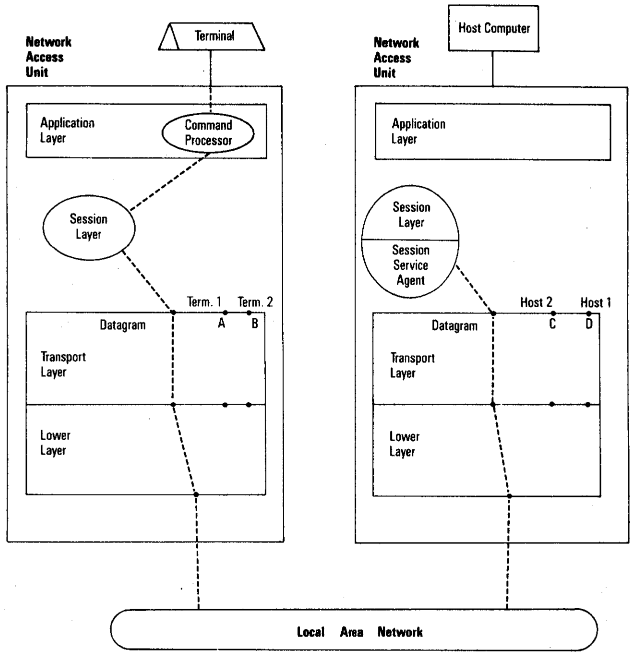


Figure 2—Data flow during remote operations

thereby protecting sensitive portions of the network from malicious users.

Another service that is useful for a session program is establishing and maintaining an array of transport sockets to which connections have been established. This is one of the features of the session layer that does not use the network protocol. The use and manipulation of the array is internal to the local network access unit, causing no network traffic, and is transparent to the resources at the other end of the connection. While the connection is active, the application resource is internally connected to the lower-layer protocol entity and sends and receives data over that connection. When the connection is interrupted or broken, the pertinent information about that connection is saved in the array. Another connection may then be established. This one may also be saved and the first one resumed. This would give the effect of putting one connection on hold while performing processing on another connection. An independent disconnect request would cause the transport connection actually to terminate.

An elaborate session layer protocol that is typical of this genre of networks represents a more production-oriented approach than is used in long-haul and in experimental networks.

Presentation Layer

The presentation layer is concerned with presenting the data to the user terminal in an acceptable format. Most of the processing needed is performed by the device driver of the output port or by a common set of library subroutines in order to minimize context switching inside the network access unit.

The presentation layer will have its own validations to perform during the connection process. If the presentation layer is not able to convert from one data stream to another, the connection will be disallowed. For example, some of the requests to convert an input stream using hardware flow control lines to one using x-on, x-off flow control characters may not be possible because of inability to map out-of-band signaling methods to other signaling methods.

The presentation layer is also used to scan the data stream for a break sequence. The occurrence of the break sequence causes the presentation layer to signal the session layer to switch the internal data stream from the network back to the local command processor. The break sequence, either a special character sequence or hardware control line change, will cause the session task to save the connection in its array and internally connect the port to the application program.

Application Layer

The application layer task in LAN access units accepts character string input from a user terminal and interprets it as a command. The simpler programs that implement this function are menu processors that accept single letter commands with few parameters. These commands each cause a single particular action such as a connect or disconnect. Elaborate application programs interpret the input character string, perform line editing as necessary (such as delete character), and then parse the command according to syntax tables. These tables tell how to interpret the parameters and which routines to run. The tables also indicate required parameters that must be supplied and give default values for those not supplied. In addition to its interface to session services, the application layer should be able to gather statistics from the other layers inside the NAU and display them in a meaningful format. It should also be able to show and set the local parameters inside the access unit.

Previously discussed was one method of affecting parame-

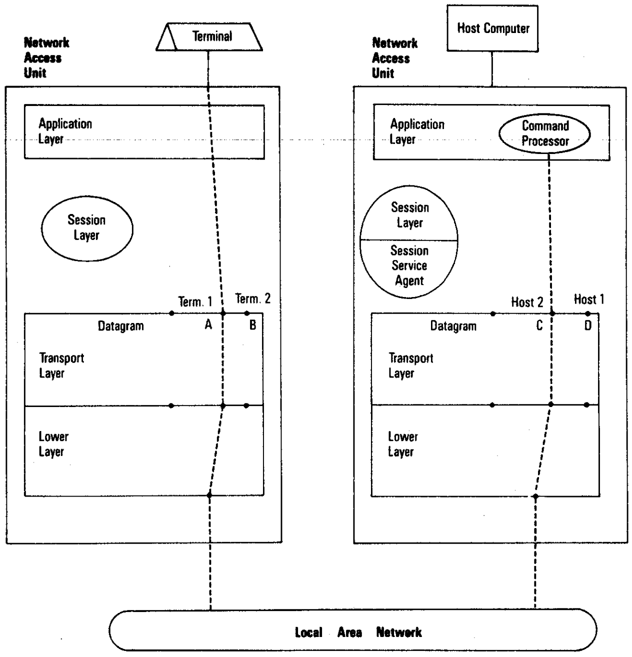


Figure 3—Data flow during remote command processing

ters on other access units, the session layer approach called remote operations. Now to be discussed is another method of accomplishing the same thing, called remote command processing. Figure 3 shows the data flow during this operation.

The application layer command processor is constructed to accept a command stream from sundry internal points. The session layer can then establish a virtual connection between a local terminal port and a remote application layer command processor. The characters entered are sent across the connection, assembled as a command by the remote access unit, and then executed. The parameters are updated or the display format generated as if the command came from a local device. In this way the access units can all be configured from a single place; there is no need to go to every unit in order to set up its parameters. This does not require the elaborate and rigorous table structures of X.409, because all requests to modify parameters are local to the NAU; they do not appear to be coming from over the network. In addition, the order and sequence of updates may be controlled locally; information transfers may be a continuous data stream and not limited to the size of a datagram.

SUMMARY

Local area networks are unique in their size and topology; this allows certain advantages to the protocol designer. Redundant operations are eliminated because processing capabilities do not allow duplicated services. The transport protocol designer can make use of the underlying layers, choosing what routines are needed to provide the services required. Network management may be performed in several ways, depending on the chosen layering and the expected use of the network.

REFERENCES

1. ISO/TC97/SC16, *Information Processing Systems—Open Systems Interconnection—Basic Reference Model,* ISO International Standard IS 7498, April 1983.
2. Clark, D. D. "Modularity and Efficiency in Protocol Implementation." *Internet Protocol Implementation Guide.* SRI International, Menlo Park, California August 1982, pp. 31–42.
3. Tanenbaum, A. S. *Computer Networks.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
4. Thurber, K. (ed.). *The Local Netter Designer's Handbook,* Architecture Technology Corporation, Minneapolis, Minnesota April 1982.
5. IEEE. *Project 802, Draft IEEE Standard 802.2, Logical Link Control.* IEEE Computer Society, Los Alamitos, California, November 1982.
6. CCITT, *Draft Recommendation X.409 Message Handling Systems: Presentation Transfer Syntax and Notation,* International Telecommunications Union, Geneva, Switzerland, 1984.
7. CCITT, *Draft Recommendation X.410, Remote Operations and Reliable Transfer Service,* International Telecommunications Union, Geneva, Switzerland, 1984.

OpenNET: A network architecture for connecting different operating systems

by LEONARD H. MAGNUSON and MICHAEL SZABADOS
Intel Corporation
Santa Clara, California

ABSTRACT

A major goal of local area networks is the connection of different systems. This paper discusses Intel's OpenNET, which provides transparent interoperation of different operating systems on the same network, based on industry standard protocols. OpenNET encompasses all seven layers of the OSI network model. The focus of this paper is on the upper-layer software, which implements the network file access capability.

A major goal of local area networks is connecting different systems. Intel's OpenNET family of products is the first to provide transparent interoperation of different operating systems on the same network, based on industry standard protocols. OpenNET is compatible with IBM's PC Networking Software and Microsoft Networks (MS-NET) and also supports Xenix and iRMX 86. OpenNET's implementation of the OSI model offers a high degree of design flexibility for system integrators and system users.

The OpenNET architecture allows transparent, heterogeneous interoperation between PC-DOS, MS-DOS, Xenix, iRMX 86, and other operating systems from vendors implementing the protocols, making possible a wide range of diverse local-area network (LAN) applications in manufacturing, transaction processing, and office automation environments. Unlike many available LAN systems, OpenNET makes use of existing file access systems, so the user's existing applications software can be used without change on the network.

The OpenNET product family consists of products that encompass all seven layers of the OSI network model. Based on Intel's advanced VLSI LAN technology, the key elements in the system are board and system-level hardware for Ethernet, transport layer software, and software that supports OSI Layers 5 through 7. Specific products in the OpenNET family include SMX 552 Multibus Transport Engine; the 186/51 Multibus COMMputer Board; iNA 960 transport layer software; and Intel's RMX Networking Software and the Xenix Networking Software, developed in a partnership between Intel and Microsoft.

The lower four layers in the PC environment are supported by the Ungerman Bass Personal NIU running a preconfigured version of the iNA 960 transport software. The upper layers for DOS operating systems are implemented by MS-NET. The products provide a set of flexible network building blocks that can be used to link different microcomputers, work stations, PCs, and peripherals. (See Figure 1.)

The focus of this paper is on the upper-layer software implementing the novel heterogeneous network file access capability on top of an ISO 8073 compatible transport capability.

The upper-layer protocols are implemented in software initially configured to run on the host processor. Layer 5 provides a logical, name-based connection capability; so systems and users can be accessed at all times, regardless of their physical location on the network. Layers 6 and 7 provide network file service. These two capabilities combine to offer transparent file access to the user from any remote or local node.

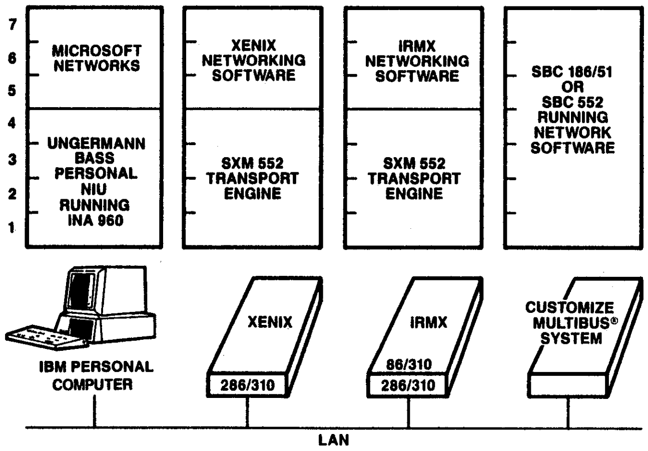


Figure 1—How Ethernet/OpenNET connects different systems on the same network

SESSION LAYER

Layer 5, the session layer, will translate the symbolic name supplied by the user into a physical address and will then use the transport layer to establish a virtual circuit between the requestor and the target system. Names are assigned by the user so that nodes can be easily accessed with familiar commands.

PRESENTATION AND APPLICATION LAYERS

The network file access (NFA) protocol lets users read, write, open, close, and otherwise manipulate files from a remote location and thus implements true transparent remote file access, as opposed to the disk-sharing and file transfer methods used by other networks. This transparency permits users to continue working with existing applications software and lets them make the transition from a stand-alone to a network environment without having to learn a new interface. On the network, each node is either a server or a consumer or both. The consumer at a given node transmits the user's commands across the LAN to the server at the access node, which executes them. Unlike file transfer (a separate capability that is not transparent to the network user), network file access generally eliminates the need to move entire files before working with the data. Network file access thus results in less movement of data along the network and higher overall network performance.

- **THE FILE ACCESS PROTOCOLS PROVIDE FOR A CONSISTENT VIEW OF FILES THROUGHOUT THE NET**

  — **THE LOCAL DIRECTORIES ARE EXTENDED ACROSS THE NET**

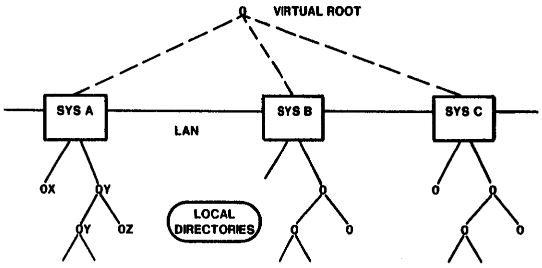


Figure 2—The network hierarchical file system

File access control is another feature of NFA. Different operating systems naturally have different protection models, and NFA honors the protection schemes specified by every type of server for its files. NFA resolves the differences between server protection models, however; so the consumer system will always see these access permissions expressed in its own operating system's format.

FILE SYSTEM

The protocols support a network hierarchical file structure, as illustrated in Figure 2. The hierarchy has been extended "upwards" through the addition of a network root (indicated by a double slash, either // or \\ to the file name space. The file names are prefixed with the name of the system on which they reside within the network. This scheme—a logical extension of the file systems used by DOS, Xenix, and iRMX 86—ensures that file names are both unique and consistent throughout the network.

User applications access remote files through the same system call interface used to access local devices. Since this interface is not modified by the network, applications transparency for remote file access is ensured. Because most services are based on the file system, many of them will work across the network with little or no change. For example, Xenix mail can exchange mail with Xenix users anywhere on the network, and the "at" command can be used to initiate batch jobs on other Xenix network systems.

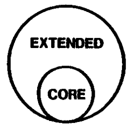
EXPANDABLE PROTOCOLS

The OpenNET file access protocols were jointly developed with Microsoft and are compatible with the protocols used by IBM in PC-NET. The application layer protocol sets are expandable to provide transparent file access among transparent interoperation between MS DOS and PC DOS systems. The extended protocols allow Xenix and iRMX-based systems to interoperate transparently. As illustrated in Figure 3, the extended protocols are a superset of the core protocols.

IMPLEMENTATIONS

The network service software consists of separate server and consumer modules. The iRMX, Xenix, and PC DOS implementations permit every node to be a server, a consumer, or both. MS DOS nodes can be either a consumer or a server.

- **THE PROTOCOLS ARE EXPANDABLE**



  — **THE CORE PROTOCOLS SUPPORT DOS-TO-DOS INTEROPERATION**

  — **THE EXTENDED PROTOCOLS SUPPORT MORE SOPHISTICATED FILE SYSTEMS (E.G. XENIX, RMX)**

Figure 3—Protocol set and heterogeneous interoperation

Implementation Under PC DOS

A PC DOS or MS DOS consumer can access iRMX, DOS, or Xenix remote files as well as remote printers connected to DOS or Xenix servers. A DOS user who wants to access a remote file or printer first connects to the file server with a NET USE command. This command equates the remote directory with a local drive identifier. Any subsequent reference to that drive identifier will automatically be routed to the redirector. A command to equate a drive identifier with a remote directory would take this form:

```
NET USE c: //System A/Directory Name
```

Once this connection is established, the user can access the remote directory by simply specifying drive c.

Implementation Under Xenix

A Xenix system may perform concurrently as both a network file consumer and a network file server. The Xenix consumer implementation offers complete file access transparency when accessing Xenix or iRMX 86 servers. Existing Xenix (or Unix) applications can generally use the network file system without change. The consumer software is embedded in the Xenix kernel and automatically directs file accesses to the appropriate local device or remote system.

The Xenix file server provides full, transparent access to PC DOS, MS DOS, iRMX 86, and Xenix systems. The server software consists of a number of kernel processes. A process is created for each remote process that makes network accesses to the server. These processes are transitory and are terminated when the remote process terminates.

Implementation Under iRMX

Like Xenix, an iRMX 86 may be configured as a file server, a file consumer, or both. When configured as an OpenNET consumer, an iRMX system may transparently access files residing on both iRMX and Xenix servers. As noted before, this implies that many iRMX applications may operate on remote files without change. If configured as an OpenNET server, an iRMX system can service PC DOS, MS DOS, Xenix, and iRMX 86 consumers.

Implementation Under Other Operating Systems

OEMs and system integrators using other operating systems and system backplanes will need customized implementations of the hardware and protocols in order to connect onto an Ethernet OpenNET LAN. The task can be considerably simplified by using the original equipment manufacturer (OEM) building blocks available from Intel.

To implement the lower four ISO layers for non-Multibus environments the OEM can develop a transport engine based on Intel's 82586, 82501,and 80186 VLSI components and the iNA 960 software.

The upper-layer software can be developed from scratch on the basis of the file access protocols, which will be published by Microsoft. In most cases, however, OEMs will save development time and resources by acquiring the source code (written in PLM for iRMX and in C for Xenix) and porting it over to their operating systems.

DMI: A PBX perspective

by THOMAS F. STOREY
AT&T Information Systems
Denver, Colorado

ABSTRACT

For the past 20 to 30 years, the Private Branch Exchange (PBX) has provided an efficient and effective means of communicating in the office environment. One of the major reasons that PBXs and the interconnecting voice-oriented network are ubiquitous, of high quality, and implementable by many vendors is the existence of technical standards defining the interface specifications between the nodes of the network. DMI is a newly proposed standard for data communications that has the potential to create for data what the existing standards have done for voice communications. In fact, much of this potential can be immediately realized, since the physical level of the DMI protocol uses the standard DS1 bit rate (1.544 Mb/s) and it is directly compatible with the high-speed T1 digital facilities existing in North America. In other words, there are billions of dollars in network facilities that DMI can leverage on immediately.

The PBX is ideally positioned to enable DMI to solve many data networking problems. The office environment is the hub of most data networking. The PBX, because of the office's voice needs, is already central to an office's voice communications network. Therefore, the digital PBX has the opportunity to efficiently solve both voice and data networking problems. For the data component, DMI is able to provide a cost-effective, high-speed interface, interconnecting terminals to hosts, and hosts to hosts on the customer's premises. In addition, the PBX is able to improve dramatically this data connectivity by extending it over a large geographic area. That is, individual 64 Kb/s DMI channels can be switched through the PBX and onto standard long-haul digital facilities. The local area network provided by DMI and the PBX can then become a wide area network that can access not only an entire country but potentially the world.

ROLE OF THE PBX

For the past 20 to 30 years, the private branch exchange (PBX) has provided an efficient and effective means of communicating in the office environment. The term PBX refers to a switching system (exchange) located on a customer's premises (private branch) that provides interconnection among terminals and facilitates access to the external network. The PBX's counterparts in the local and long-distance network are the central office switch and the toll trunk switcher. These switches, together with the interconnecting facilities, have created a network in the U.S. that is not only a technological marvel but the envy of the world. From any PBX network terminal a high-quality, voice-oriented connection can be set up instantly to more than 200 million end points anywhere in the country.

The network, which has been optimized for voice connections, also has been used extensively to transport data between terminals and hosts as well as between hosts. Modems are used to convert the per-channel digital bit streams from these devices to analog signals that permit the transmission of the data onto a connection that would normally carry voice. However, with the rapid growth of PBX data end points in recent years, and the limitations created with modems, it is desirable to transmit the digital bit streams between the terminal and host end points transparently within the PBX. It is also desirable to access the network directly with the digital bit streams. Digital PBXs have made this possible.

A digital PBX uses a digital network fabric. That is, it performs its connection function by switching the digital bit stream received directly from each originating end point to each terminating end point. Digital switching in the PBX has many advantages, including the ability to increase the transmission rates between the end points dramatically and cost effectively. The latter also applies to the network access interface.

EFFECT OF STANDARDS ON THE VOICE NETWORK

One major reason that the existing voice-oriented network is widespread, of high quality, and implementable by many vendors is the existence of technical standards that define the interface specifications between the nodes of the network. AT&T has had a history of creating network technical standards and making them available to all so that there can be one network. In particular, the United States Independent Telephone Association (USITA) and AT&T have collectively contributed over the years to the "Notes on Distance Dialing," which have provided technical data required by designers and engineers to design and plan the existing network in areas such as the numbering plan, the switching plan, equipment, signaling, and in network management, transmission, and maintenance.

DMI DATA NETWORKING STANDARD

The digital multiplexed interface (DMI) is a proposed standard for data communications that has the potential to create for data what the existing standards have done for voice communications. In fact, much of this potential can be realized immediately, because the physical level of the DMI protocol uses the standard DS1 bit rate (1.544 Mb/s) and it is directly compatible with North American high-speed T1 digital facilities. In other words, there are billions of dollars' worth of network facilities that DMI can use immediately.

DMI also is consistent with international standards; the North American standard is used in Japan. The international telecommunications organization, CCITT, is actively considering the Integrated Services Digital Network (ISDN) standards proposals. DMI has incorporated the ISDN standards wherever possible. At the physical level, DMI supports both North American (1.544 Mb/s) and European (2.048 Mb/s) high-speed digital facility line rates. This provides 24 and 32 64-Kb/s channels, respectively. For the message base control channel the protocol is the same as the CCITT standard, and it is hoped that CCITT will adopt the DMI mode 3 protocol for the ISDN data channels.

The PBX is ideally positioned to enable DMI to solve many data networking problems. That is, the office environment is the hub of most data networking. The PBX, because of the office's voice needs, is already central to most office communications networks. Therefore, the digital PBX has the opportunity to solve both voice and data networking problems. For the data component, DMI is able to provide a cost-effective, high-speed interface, connecting terminals to hosts and hosts to hosts on the customer's premises. In addition, the PBX is able to improve dramatically this data connection by extending it over a large geographic area. That is, individual 64-Kb/s DMI channels can be switched through the PBX and onto standard long-haul digital facilities. The local area network provided by DMI and the PBX can then become a wide area network that can access not only an entire country but potentially the world. For the existing voice implementation of the network, the features and capabilities are already well established. With the PBX providing the networking for data as well as voice, several significant benefits can also be realized including a common network and control structure providing common maintenance and administration, a common access to both the work station and the network, and the opportunity to provide integrated voice–data services (because simultaneous access is available to both).

PBX ARCHITECTURE

The PBX provides a reliable, easily configurable switching network fabric that enables interconnection of voice or data terminals within the office. It also permits the interconnection of PBX end points to the network. The PBX makes the interconnections by setting up a two-way circuit for the duration of a call between the two end points involved. Up until a few years ago, the circuit switch connection transmitted the analog representation of the voice or data connection. Today, the PBX uses a digital representation of the signal from the end points. The switch fabric therefore transmits a digital bit stream. For voice-oriented end points the analog signal is sampled at 8 KHz and each sample is represented by an 8-bit code. It therefore takes 64 Kb/s to represent a voice connection. For data end points, a 64-Kb/s switched data channel is not only a basic switchable channel but it is easy to implement. The PBX digital swtich fabric also can be configured for simultaneous switching of very high bandwidth channels that are multiples of 64-Kb/s channels (i.e., 384 Kb/s or 1.544 Mb/s). Each physical end point in a PBX environment normally has two terminals, one for voice and one for data. It is therefore advantageous to have at least 128 Kb/s and a control channel at every end point.

Data networking with DMI therefore has a major advantage to draw on, the PBX itself. The modern PBX can improve the implementation of a cost-effective, widespread, and reliable data network for the office. It also can provide connection to the rest of the world. The following sections outline some of the attributes of a modern PBX and the design technology that should enable the PBX and its DMI interface to achieve the attributes stated above.

PBX Switching Technology

PBX switching systems have a rich design base and a history of using the most up-to-date hardware and software to create automated and reconfigurable networks that connect thousands of end points. These systems operate in real time, are capable of high performance, often are fault tolerant, and are designed to meet the needs of the office environment. AT&T Information Systems' System 85 implements several capabilities (Figure 1).

The system uses a high-performance, bit-sliced, common control processor (501CC) that can be duplicated for reliability. The duplicated processor, a cache memory, many megawords of random access memory, and a nonvolatile bulk storage subsystem all have built-in error detection and, in some cases, error correction. The processor and its subsystems can be reconfigured automatically in real time, without loss of service. In addition, the entire system can be diagnosed to the circuit pack (replaceable entity) level, again without service interruption. The entire duplicated system has a downtime objective of three minutes per year.

The switching network consists of two basic building blocks: a time slot interchange (TSI) module and a center-stage time multiplex switch (TMS). The TSI switches digital bit streams arriving from terminal end points by reordering the time sequence of the bit streams received. The reordered digital bit streams are then sequenced onto the digital highways to

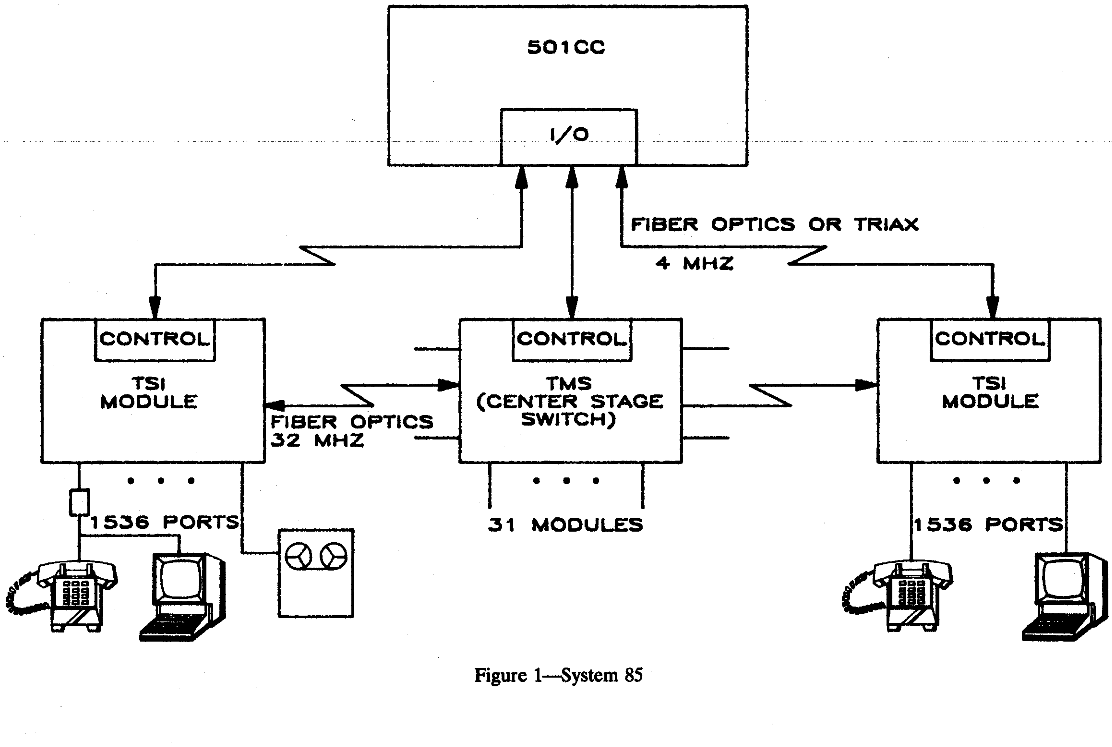


Figure 1—System 85

return to the appropriate terminal end points. The TMS connects modules with one another and performs a space-switching function by directly connecting up to 31 TSI modules at any given time. Because this space-switching function is multiplexed, the TMS is reconfigured millions of times each second.

Individual terminals are connected by twisted pair cables to port circuits in a TSI module that can be 5000 feet away. For normal digital end points these twisted pair cables carry data at 160 Kb/s. Port circuits format the digital bit streams and place them on the high-speed digital highways terminating on the TSI. The TSI has a switching capacity of 65 Mb/s. The capacity of the TMS is 1024 Mb/s. The TSIs are connected to the TMS by a 32-Mb/s fiber optic link that can be 13,000 feet long.

The digital switching network fabric has the same fault-tolerant capabilities as the processor complex. All the TSIs, the TMS, and their interconnecting fiber optic links can be duplicated. Built-in error detection, error recovery, and diagnostics permit automatic reconfiguration under error conditions. They also allow easy location and replacement of failing circuit entities. For routine maintenance of redundant units, the system is even able to switch from the on-line to off-line network modules without missing a single bit passing through the network.

In addition to the 32-Mb/s fiber optic links, the system takes advantage of the latest in high-speed logic and both custom and off-the-shelf VLSI devices. For example, 256-K memory parts are used in the common control, custom VLSI in the port circuits, and high-speed ECL logic in the TMS.

BENEFITS OF PBX VOICE–DATA INTEGRATON

Because the PBX has a large installed base that currently implements a multiplicity of voice functions it provides an opportunity not only to leverage data implementations from the existing voice network but also to develop synergies between voice and data functions on the PBX.

Installed PBX Base

Digital PBXs are being installed at a rapid rate, about 10,000 systems in 1985 in the U.S. alone. DMI allows efficient connection of host computers to these PBXs and in turn to the many terminal end points attached to these same PBXs.

Common Access to Long-Distance Network

The PBX provides an efficient means of connection to the long-distance network. PBXs are directly attached to the network, often with digital facilities. This access and the transmission over long distances is expensive. DMI data channels can share access and transmission costs with the digitized voice channels that traverse this interface.

Installed Network Facility Base

The long-distance network is rapidly becoming digital. DMI is directly compatible with this vast resource because the physical layer of the DMI protocol is identical to the North American T1 digital facility standard.

Installed Building Wiring Base

In most offices, huge investments have already been made in twisted pair copper wire that connects telephone station sets to PBXs, both old and new. This same twisted pair wire can be used to transmit high-speed digital signals from work stations to the PBX.

Voice–Data Feature Integration

The ability to use shared resources for voice and data also offers the opportunity to develop features that take advantage of the simultaneous access to voice and data information.

ISDN Directions

DMI is directly compatible with the ISDN standards. In addition, it should not be long before ISDN network standards are accepted for PBX-to-PBX and PBX-to-network interfaces, not only in the U.S., but worldwide. When this happens, DMI's advantages will multiply for many of the reasons outlined above.

DMI ARCHITECTURE AND DESIGN IN THE PBX

To outline the architecture of the DMI interface in a PBX, the implementation of System 85 will be highlighted. System 75's implementation is similar. Figure 2 shows a system level view of the connection between terminal end points and host computers and between one host computer and another. It also illustrates the interface to the network.

PBX to Host Computer

DMI defines the PBX-to-host-computer interface. For domestic implementation, the physical level is a 1.544-Mb/s bit stream that uses the standard North American T1 framing formats and provides 23 64-Kb/s data channels and one 64-Kb/s common signaling channel. The output of a DMI port on either the PBX or the host computer can interface directly to external facilities (no digital channel banks), providing an ability to use this interface easily at long distances (thousands of miles) over digital facilities.

For a direct interface to the external network, the FCC requires the use of a network channel terminating equipment (NCTE) device. The NCTE terminates the repeatered T1 line, provides T1 maintenance capabilities, and provides a DS1-compatible interface to the PBX or host computer. The maximum distance from the NCTE to the PBX or the host is 655 feet over twisted pair cable. A special twisted pair shielded cable is required for this interface.

For use in an office, the DS1 metallic interface can extend 1310 feet between the PBX and the host. Again, the special

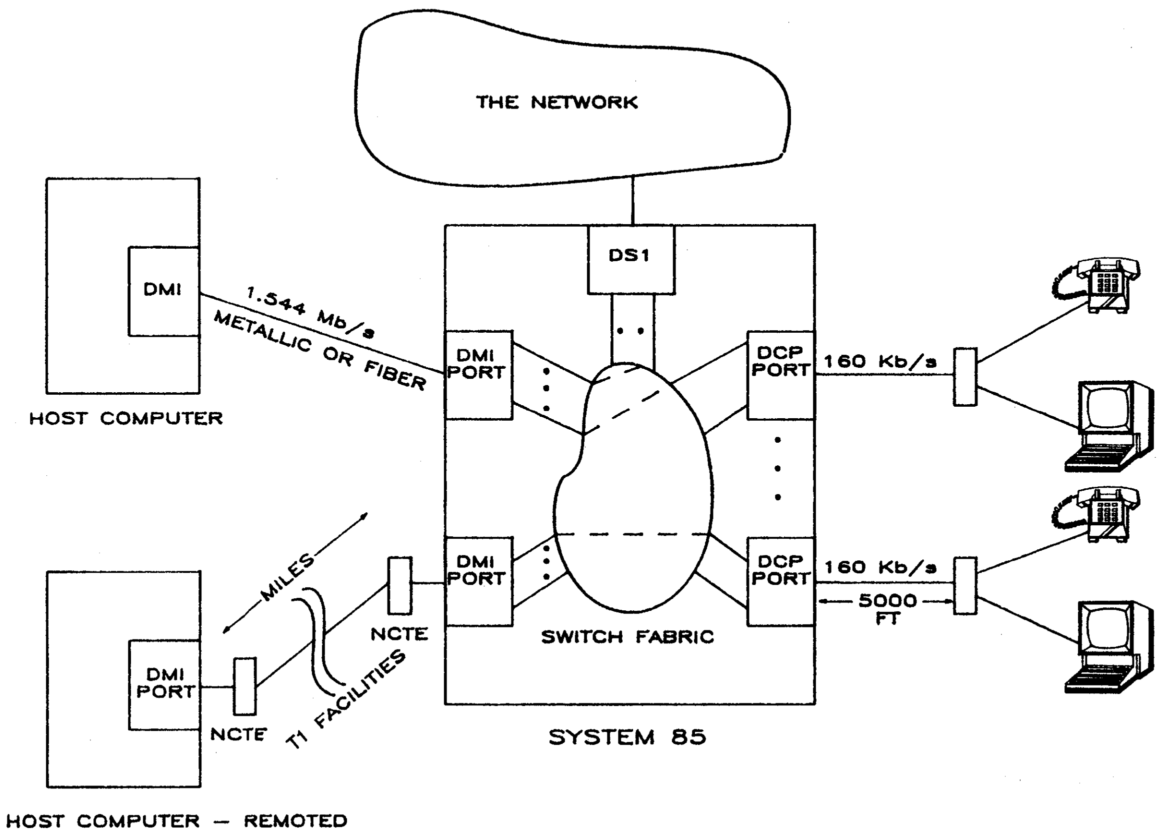


Figure 2—System level view

twisted pair shielded cable is required. A fiber optic interface option from the DMI port will permit DMI to extend 2-km for large buildings or complexes.

Figure 3 is a detailed block diagram of the DMI port design in the PBX. The entire interface is contained on a single, 8- by 14-inch circuit pack. The major sections of the DMI interface are the DS1 1.544-Mb/s line interface (implemented with transformers and a VLSI DS1 chip set), the fiber optic link interface, the interface to the high-speed digital bit stream highways that connect the DMI port to the TSI switch entity, the interface to common control of the digital switch fabric, and a microprocessor control complex to provide a variety of functions on the DMI port. The microprocessor functions include processing the signaling information in the 24th channel, providing the per-channel control association with each data channel, initiating start-up, and setting default parameters for the port. The microprocessor also performs error monitoring and maintenance testing both in real time and on demand for many functions both on this port and on the connected facilities.

For the metallic implementation, the DMI 1.544-Mb/s bit stream arrives at the receive input. Passing through the receive transformer, it then encounters in sequence the receive converter, the framer, and the receive synchronizer. These devices convert the T1 alternate mark inversion (AMI) signal into a parallel data format, recover clock from the incoming bit stream, and identify the DS1 frame boundaries. These devices also double-buffer the incoming data streams to eliminate any loss of data that might occur because of phase jitter between the DS1 line clock and the PBX network clock. In addition, they provide a 32, 64-Kb/s-channel format for the data that are transmitted to the switch network. The 32-channel interface to the switch allows the same basic design to be used both for domestic (24) and for international (32) facility interfaces. In the transmit direction the inverse functions are performed. The maintenance buffer device provides an 8-bit microprocessor bus on one side and a serial channel interface to the DS1 chip set on the other. The serial channel provides the access and the capability to run both in-serivce and out-of-service error detection and maintenance routines on the DMI port. The maintenance buffer also monitors in-service facility alarms and records the number of misframes and slips on the DMI facility.

The fiber optic interface operates in the same manner as the metallic interface except that a fiber optic transmitter and receiver replace the transformers and the transmitter and receive converters, respectively.

In the DMI interface at the PBX, the level two and level three functions of the DMI protocol are not performed. The HDLC data streams are transmitted directly through the DMI port and the switching network to the terminal or host computer end points, where these levels of the protocol are terminated. For PBX terminals that are not compatible with DMI protocols, an adapter or protocol converter is needed. This

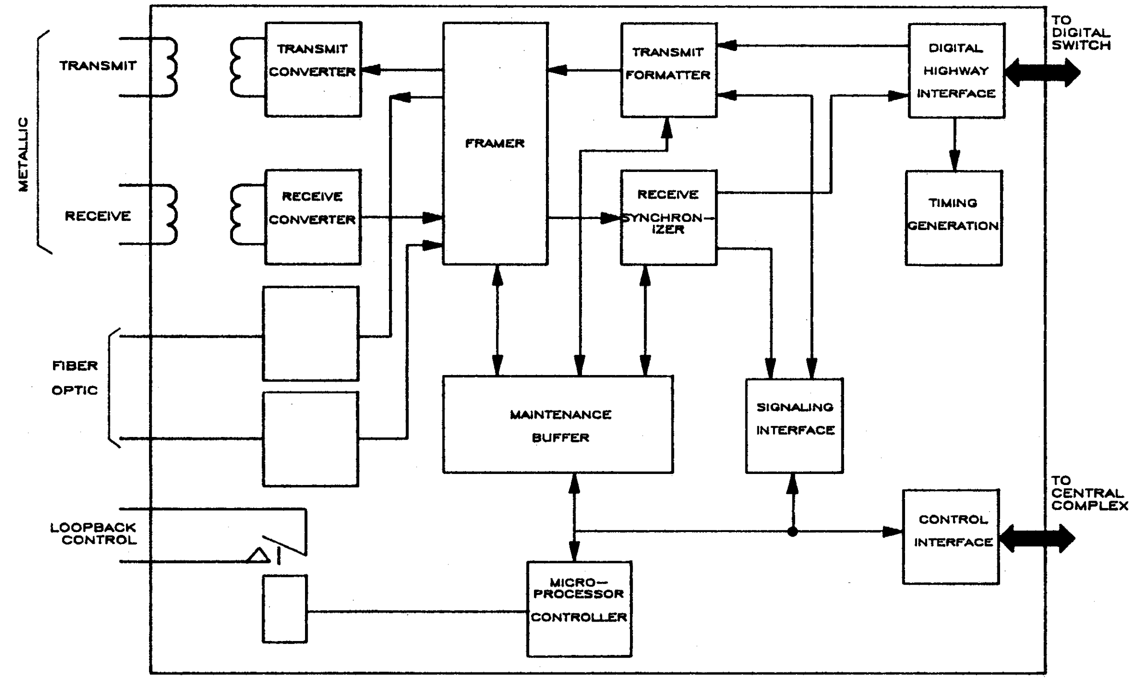


Figure 3—System 85 DS1-DMI port

adapter can be put in series with the DMI interface or it can be pooled and switched into DMI connections as shown in Figure 4.

PBX to Network

The same physical interface described for the DMI host port in the PBX can also be used for the PBX port that interfaces to the network, except for the firmware that controls the details of the network signaling and sequencing patterns.

PBX to Terminal

The PBX-to-terminal interface probably will be a little different for each PBX vendor. However, there will typically be

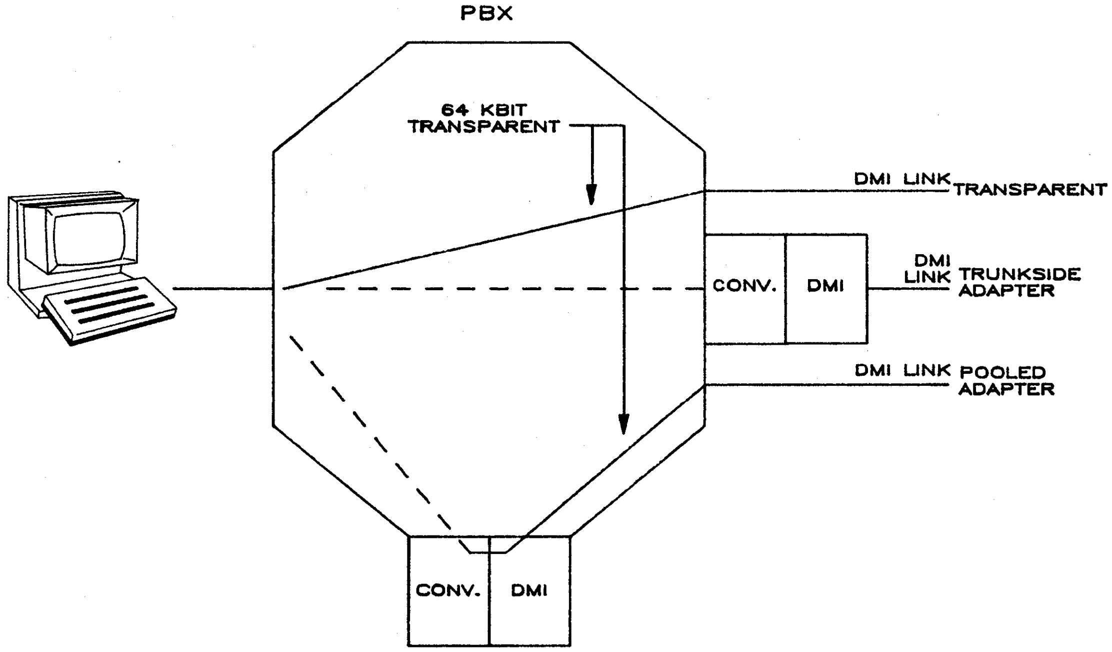


Figure 4—DMI protocol conversion alternatives

some areas of commonality: one or two 64-Kb/s data channels for each work station, a moderate speed control channel (8- or 16-Kb/s) and a physical line interface that allows these channels and direct current power to be multiplexed onto twisted pair copper wires and transmitted thousands of feet from the PBX switch.

In System 75/85, the digital terminal line interface is defined with a protocol called DCP (digital communications protocol), which initially was chosen to be compatible with the ISDN terminal interface. It is therefore similar to the ISDN standard, but in the interval since the DCP design was completed a few years ago, the ISDN standards have been modified. The DCP line rate is 160 Kb/s; two 64-Kb/s channels (one for voice and one for data), 8 Kb/s for control, with the remaining bandwidth for framing. The DCP line can support digital end points up to 5000 feet from the PBX. DCP supports only point-to-point line terminations.

COST IS THE BENEFIT

The multiplexed nature of the DMI interface between the PBX and the host computer will provide a major cost improvement compared with current methods of interconnection. Individual lines and the associated interface hardware now are required for every connection, both at the PBX and at the host end. Typically it will be more cost effective to use a 23-channel DMI link rather than four or five individual links using modems and per-channel hardware. Therefore, the implementation of large numbers of terminals or personal computers connecting to a host will be cost efficient.

DMI also has many feature attributes that will greatly enhance the utility of the interface:

1. DMI offers a direct interface to standard T1 facilities, which "remote" the PBX-to-host multiplexed connection.
2. DMI also provides the ability to connect equipment from several vendors to one standard interface.
3. DMI protocol options offer a range of HDLC-compatible alternatives for transmition of high-speed digital data over a large embedded base of digital facilities. DMI mode 3 will be particularly attractive because of its error detection and error correction capability, its ability to implement statistical multiplexing on DMI links, and the simplicity and cost effectiveness of terminating the protocol at terminal end points.

SUMMARY

DMI has been proposed by AT&T Information Systems as a standard to help create a universal interface that will allow efficient networking of computers through PBXs. The large embedded base of digital PBXs and the vast long-distance network to which PBXs have access, make the PBX an extremely powerful networking vehicle. Today's PBX can routinely switch 64-Kb/s channels; many can switch Mb/s channels if so required.

AT&T Information Systems and Hewlett-Packard have been working together for the past year to ensure a DMI interface that can be implemented universally. This has included working with Rockwell International and AT&T Technologies to ensure the availability of VLSI support to facilitate a straightforward and cost-effective design of the DMI interfaces both at the PBX and the host computer. Design implementation details and VLSI descriptions are provided to all DMI licensees at the DMI User's Group Meetings held every few months. It is expected that the availability, the technical content, and the many capabilities of the DMI specification, together with VLSI device support and the implementation details provided, will enable DMI to become a universal data communications standard. This standard should benefit all PBX and host computer vendors. Most important, it should soon provide the users with a class of data service that rivals the voice services offered today by the PBX and the long-distance network.

Digital Multiplexed Interface: Architecture and specifications

by JAMES L. NEIGH
AT&T Information Systems
Lincroft, New Jersey

ABSTRACT

The Digital Multiplexed Interface provides cost-effective data connectivity between PBX-based terminals and host computers. DMI uses high-speed digital transmission facilities, common channel signaling, and four data transport modes to provide data connectivity at rates up to 64 Kbps for 23 (30 in the European version) data channels over a single 4-wire interface. DMI is consistent with both the existing and evolving standards for the Integrated Services Digital Network and is probably the first practical example of implementation of an ISDN interface. Since DMI was initially announced in November 1983, over 60 companies have licensed the right to build DMI, and many host and PBX vendors are currently implementing the interface. DMI licensees are aided in their implementations through participation in the DMI Users Group. The Users Group includes detailed information and discussion on DMI implementation, in addition to information on standards activity related to DMI and discussions of plans for continued DMI evolution. DMI's capabilities were demonstrated between an AT&T Information Systems PBX and a Hewlett-Packard 3000 host computer at Interface '85 in Atlanta. PBX and host equipment with DMI interfaces will begin being shipped to customer locations in the second half of 1985.

INTRODUCTION

The Digital Multiplexed Interface (DMI) was defined to provide cost-effective, high-speed interconnection between terminals and host computers in a PBX environment. The DMI definition maintains a balance between maximum use of existing technologies to facilitate early implementations and compatibility with emerging international standards for the Integrated Services Digital Network. DMI takes advantage of existing high-speed digital carrier systems (1.544 Mbps in North America and Japan; 2.048 Mbps in Europe) to provide multiple (23 or 30) 64-Kbps data communications channels over a single physical interface. To provide a smooth evolution to ISDN, DMI uses existing AT&T Information Systems algorithms for data transport over a 64-Kbps channel and confines signaling to a single common channel.

The initial public announcement of support for DMI was made jointly by AT&T Information Systems, Hewlett-Packard Company, and Wang Laboratories in October 1983. Since that time many additional vendors have licensed for the right to develop DMI on their products. This group of licensees includes most of the significant vendors in today's voice/data communications marketplace.

PRE-DMI DATA CONNECTIVITY

Without a DMI-like interface, data communication between a PBX and host computer is limited in several respects. The most common arrangement requires separate wire pairs, a physical termination, and a pair of modems for each data channel. In addition to the obvious cost burdens of this arrangement, functionality is limited by a maximum data rate of 19.2 Kbps.

The economic burden can be eased in some applications through the use of statistical multiplexers to achieve more efficient use of the physical connections. The maximum data rates can be increased through the use of product specific data modules rather than modems. However, none of the existing arrangements provide an efficient means for high-speed multiplexed data transport between a PBX and a host.

Figure 1 shows three alternatives for data transport between a PBX and host: (1) the use of modems on a per-channel basis, (2) product-specific data modules that provide two 64-Kbps data channels per termination, and (3) DMI. Figure 2 shows a relative cost comparison for one example of an intrapremise implementation of each of the three alternatives. As shown in the figure, DMI is more cost-effective than the 2-channel digital interface when three or more data channels are required. Similarly, when compared to the per-channel data modem alternative, DMI is economically attractive when five or more data channels are required. Thus, DMI is an attractive solution even when relatively modest levels of data connectivity are required between PBX-based terminals and the host.

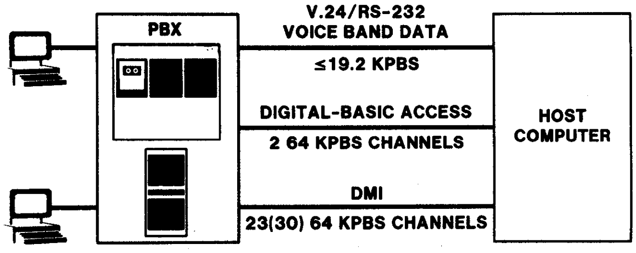


Figure 1—Data connectivity alternatives

DMI DESCRIPTION

Overview

The North American version of DMI is based on use of the standard DS1 (T-carrier) interface. The DS1 interface is 4-wire and supports a 1.544-Mbps digital signal. The 1.544-Mbps signal is channelized into 24 64-Kbps channels plus one 8-Kbps channel for framing. Channels 1 through 23 are used as information channels; channel 24 is a common signaling channel. Each of the information channels can support standard data rates up to 64 Kbps. In initial applications, the common signaling channel will make use of multiplexed *A*-bits to indicate on and off hook and dial pulse information. Present plans call for implementation of a message-oriented signaling protocol, which is a subset of the protocol defined by the CCITT for customer access to the Integrated Services Digital Network approximately one year after the initial release of DMI.

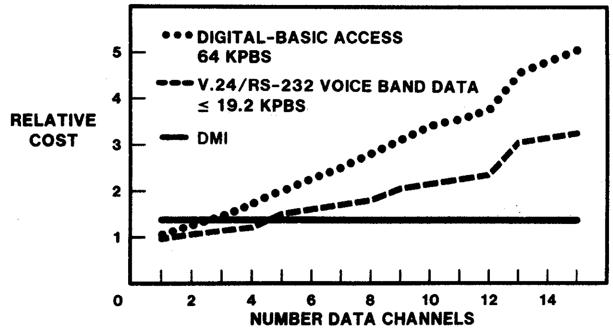


Figure 2—Cost comparison: DMI vs. alternative data transport capabilities

The European version of DMI is based on the CEPT standard 2.048-Mbps digital interface. This interface is divided into 32 64-Kbps channels: 30 information channels, a common signaling channel, and a framing channel. Data transport formats are identical for European and North American versions of DMI. The common signaling channel in initial implementations uses the CEPT standard for multiplexed A-bits in channel 16. Later versions will use the CCITT standard message-oriented signaling.

Connectivity

The use of standard digital interfaces permits almost unlimited flexibility in the distances over which DMI can be used. For local applications, DMI can operate at distances up to 1300 feet between the PBX and host computer without signal regeneration. For distances greater than 1300 feet, DMI makes use of the standard T-carrier repeater line, or a fiber optic option. The T-carrier facility might be installed on a private basis or leased from a communications provider for long-haul applications. Thus, DMI could be used to link a PBX and host computer located as close as the same equipment room or as far as opposite coasts of the United States.

In addition to the essentially unlimited distance capability of DMI, the use of standard 64-Kbps channels permits the additional flexibility of switching the individual data channels from one interface to another within a digital PBX. Thus, for PBXs interconnected by DSI facilities, a DMI host at one location can establish a direct 64-Kbps path through its PBX to a terminal or host at a different location. An example configuration is shown in Figure 3. In this example, if the terminals and hosts support common DMI data transport modes and the PBXs support common channel signaling on the inter-PBX links, full connectivity between any pair of terminals and hosts is possible with no intermediate protocol conversions required.

Physical Interface

The DMI physical interface is based completely on North American, European, and CCITT standards for the use of digital transmission facilities at 1.544 and 2.048 Mbps. Implementation of the physical interface is identical to implementations on existing digital switching equipment.

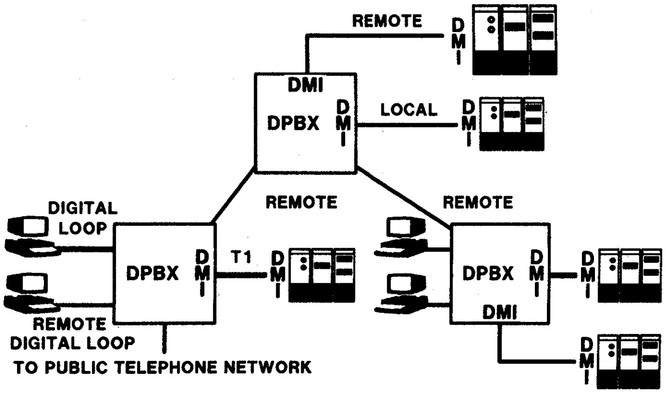


Figure 3—DMI application example

Key characteristics of the physical interface are listed in Table 1. Also shown is the identification of the CCITT recommendation which defines each of the characteristics.

Signaling

To facilitate early implementations, DMI will support two types of common channel signaling. For initial implementations, and situations in which a sophisticated signaling capability is not required, bit-oriented signaling (BOS) will be used. With BOS, a single bit (A-bit) is assigned to each data channel. The A-bit is used to indicate the state (on hook, A = 0; off hook, A = 1) of its corresponding data channel, and to convey address information. The A-bits are multiplexed in the common channel by assigning the A-bits for consecutive channels to successive frames of the digital signal. In the simplest application of the North American interface, the A-bits for each channel are updated once every 24 frames.

In order to support increased functionality and to establish a clear evolution to ISDN, the initial release of DMI will be closely followed by a version with Message Oriented Signaling (MOS). DMI's message-oriented signaling is a fully compatible subset of CCITT recommendations I.440/Q.921 and I.450/Q.931. These recommendations define the Layer-2 and -3 signaling protocols for customer access to ISDN. A few examples of enhancements provided by message oriented signaling are faster call setup, calling number identification, and the ability to send user-to-user information.

Data Transport

Four modes, 0, 1, 2, and 3, have been defined for data transport on DMI. Mode 0 is for clear 64-Kbps data transport; in this case, the end user defines the higher layers of the data transport protocols. When defining mode-0 protocols for applications which may include transmission over North American T-carrier facilities, care must be taken to ensure that the data channels meet T-carrier 1's density constraints.

Mode 1 defines unconstrained use of a 56-Kbps data channel. Mode 1 is included in DMI to provide compatibility with Digital Data Service facilities.

Mode 2 is defined to provide transport of synchronous and asynchronous data at standard rates up to 19.2 Kbps. Mode 2's principal application is support of existing terminal equip-

TABLE I—DMI Physical Interface

	NORTH AMERICA/JAPAN	EUROPE
RATE	1.544 MBPS	2.048 MBPS
NUMBER DATA CHANNELS	23	30
FRAMING	CCITT G.733 (i.e. D4 AND F_e)	CCITT G.734
ELECTRICAL CHARACTERISTICS	CCITT G.703	CCITT G.703
CLEAR CHANNEL	B8ZS	HDB3

ment using RS232- or V.24-type interfaces. Mode 2 uses an HDLC-based frame approach to rate-adapt the existing data rates to 64 Kbps. The Mode 2 definition also includes a handshaking procedure at call initiation to establish various parameters such as speed, full/half duplex, and timing options. Mode 2 also includes an update message which indicates the status of the interface control leads at the remote interface.

DMI Mode 3 provides for statistical multiplexing of multiple data channels onto a single 64-Kbps channel. In addition, each of the multiplexed channels provides reliable data transmission because link layer procedures for sequencing, retransmission, and flow control have been defined. Mode 3 is based on CCITT recommendation I.440/Q.921, which was defined to provide link layer multiplexing of signaling and low-speed data on the ISDN signaling (D) channel.

DMI SPECIFICATION

The DMI specification contains a detailed description of the interface. The specification is separated into four basic parts: (1) an overview/introduction, (2) the physical interface description, (3) the common channel signaling protocols, and (4) the data transport formats. Issue 1.0 of the specification was made available to the general public at the March 13, 1984, meeting of the Electronic Industries Association TR41.1. Issue 1.0 was incomplete in that the message-oriented signaling protocols were not fully specified and the Mode 3 data transport format had not yet been introduced. Since the release of Issue 1.0, the DMI specification has been significantly enhanced through the introduction of new technical material and the addition of clarifying text. The present issue of the specification, Issue 3.0, presents a mature view of the interface. No significant changes to the existing sections of Issue 3.0 are anticipated. However, additions to the specification may be made to introduce enhancements which are a natural part of any evolving interface.

IMPLEMENTATION SUPPORT

DMI licensees are supported in their development of the interface by the availability of devices for DMI implementation and by membership in the DMI users group. Both Rockwell International and AT&T Technologies provide devices for implementation of the DS1 physical interface, and for the per-channel HDLC handling that is required for implementation of Mode 2 and Mode 3 data transport. The Rockwell DS1 devices are part of their standard DS1 chip set and are currently available. The Rockwell per-channel HDLC device, called the Highly Integrated HDLC Interface (HIHI), is under development. HIHI will provide 24 or 30 HDLC interfaces on a single chip. It is being developed specifically for DMI implementation. Initial availability is expected by the end of 1985.

The AT&T Technologies DS1 chip set is also currently available for DMI implementation. The AT&T Technologies HDLC device is called the Spyder. It is a general-purpose HDLC device with eight channels per chip. First availability of Spyder is also expected by the end of 1985.

The DMI user group was initiated to aid licensees in their implementation of DMI. A typical user group meeting will provide licensees with a status report regarding latest developments on the interface, explain technical details of the DMI specification, discuss implementation issues ranging from devices to certification testing, and answer any licensee questions. User group meetings have been held approximately every 4 months and have received a very favorable reaction from the licensees.

ISDN STANDARDS

DMI has been proposed as an industry standard for PBX to host data communication. In defining DMI, care was taken to ensure that DMI is consistent with the evolving standards for the Integrated Services Digital Network. The first set of formal recommendations for ISDN were approved at the CCITT Plenary Assembly in October 1984. The aspects of ISDN included in DMI are the primary interface rate, the DS1 physical interface, message-oriented common channel signaling, and the support of 64-Kbps clear channels. As such, DMI is a member of the ISDN interface family, and implementation of DMI is both consistent and compatible with the support of ISDN interfaces to both public and private networks.

In addition to adopting appropriate CCITT Recommendations in DMI, various aspects of the interface have been discussed within the Electronic Industries Association and the European Computer Manufacturers Association. Examples of ongoing activity of significance to DMI includes an ECMA effort to establish ISDN-based standards for interfaces between Data Processing Equipment and Private Circuit Switched Networks and CCITT efforts to establish a packet-oriented data transport protocol for use on ISDN bearer channels. The ECMA draft standard for a Q.931-based signaling interface is essentially equivalent to Message Oriented Signaling on DMI. Because of its outgrowth from the ISDN D-channel protocol (LAPD), one candidate for packet mode transport on ISDN bearer channels is a generalization of DMI Mode 3.

SUMMARY

DMI provides a cost-effective interface for multiplexed PBX to host data communication. Because it is based on the use of standard T-carrier facilities and 64-Kbps transmission and switching, applications of DMI can provide data connectivity over geographically dispersed private networks. DMI has been defined to provide an implementable interface for today's applications and to be consistent with the evolving standards for the Integrated Services Digital Network.

Many major suppliers in the voice/data communications marketplace have licensed DMI and are building the interface on their forward-looking products. Implementation support is provided to these vendors by encouraging the availability of appropriate devices and through formation of the DMI Users Group. Implementation of DMI is an opportunity to provide a cost-effective interface providing a clear path for product evolution toward ISDN.

Digital Multiplexed Interface: A host side implementation

by TIMOTHY C. SHAFER
Hewlett-Packard Company
Roseville, California

ABSTRACT

The CCITT's Integrated Services Digital Network standards are the foundation for a new generation of PBX data networking products. Hewlett-Packard Company, AT&T Information Systems, and other leading computer and PBX manufacturers have taken the first step toward economical ISDN interfacing through the development of products based on the Digital Multiplexed Interface specification (DMI). DMI has a two-phase architecture that allows development of an ISDN-compatible, high-bandwidth, multichannel computer-to-PBX interface using T1 or CEPT carriers.

INTRODUCTION

Moving data communications onto telephone networks is becoming increasingly important to office automation. The large installed base of telephone lines, the rising cost of installing new cable, and the growing importance of data communications for business are inspiring computer manufacturers to build new interface products. These products are inexpensive, standard interfaces that connect computers to PBX and other telephone switching equipment. Applying these interfaces to office automation equipment will change the role of digital PBX in the office by making it the workhorse for most data communication. The international telecommunication organization, International Consultive Committee for Telephone and Telegraph (CCITT) recently approved Integrated Services Digital network (ISDN) standards. This is an international acknowledgment of the growing importance of moving data onto telephone networks through the development of a standard set of interfaces and services. The ISDN recommendations provide an architecture that manufacturers in the PBX and computer industries can use as a basis for development of systems to meet the need for open-system architecture.

AT&T Information Systems (AT&TIS), Hewlett-Packard Company (HP), Wang Laboratories, and other leading computer and PBX system manufacturers have taken the first step toward ISDN networking through the development of products based on AT&T's digital multiplexed interface (DMI) specification. Founded on the ISDN Primary Rate interface standard, DMI was developed by AT&TIS as an open interface for connecting host computers to PBXs over a T1 (or CEPT, in Europe) carrier. DMI has a two-phase architecture: The first phase specifies the development of an interface that is similar to the ISDN Primary Rate but simpler to implement; the second specifies exact implementation of the Primary Rate. This approach was taken because the ISDN standards were evolving and there was an immediate need for an interim interface to provide substantial savings over existing PBX-based solutions. Another reason for this approach was to find a system that could migrate smoothly and economically to ISDN compatibility.

Hewlett-Packard's development of DMI and the importance of DMI as a starting point for the development of ISDN-compatible interfaces will be considered in this paper. Manufacturers implementing the DMI specification will replace the individual single-channel data lines and data modules that are currently used for PBX networking with the high-bandwidth, multichannel DMI interface (Figure 1). Our initial DMI product attaches the HP 3000 computer to AT&T's System 75/85 PBX and to the systems of other PBX manufacturers—this implements the first phase. We chose to use DMI because the interface was close enough to the ISDN Primary Rate standard that we believed the product could be built with the capability of being upgraded to ISDN conformance via firmware. Since we began development, DMI has gained considerable support from many PBX and computer manufacturers, reaffirming our belief that it is an excellent initial approach to ISDN networking.

SETTING DEVELOPMENT GOALS

The HP DMI development effort concentrated on modularity, leveraging our experience with other serial interface products, existing integrated circuits, and flexibility in protocol handling. Modularity and foresight in drawing the boundary between hardware and software implementation were important to building in a smooth upgrade capability. Applying our experience with terminal multiplexing products and existing integrated circuits allowed us to begin implementation quickly and to target a cost that allowed the product to be priced attractively. Our target was to provide an approximately 25% reduction in cost for customers (using average industry prices) compared with the price of several single-channel lines and proprietary data modules at the host interface. Our goal, flexibility in protocol handling, was necessary because there are different formats used by ISDN for transporting data.

Our initial product is an implementation of the first phase of the DMI specification; the fixed, simplified version of the ISDN Primary Rate interface standard. The product is a 24-

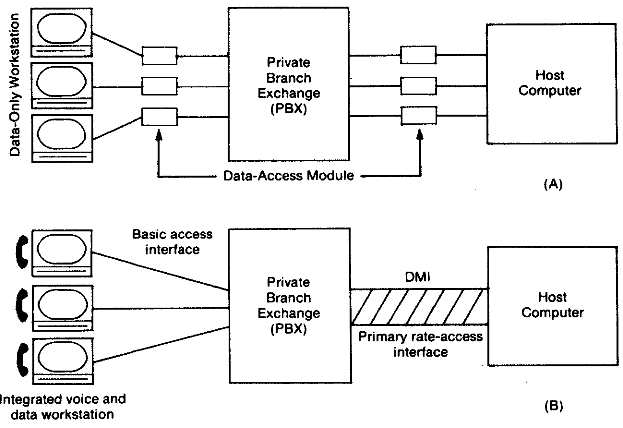

Data-Only Workstation
Private Branch Exchange (PBX)
Host Computer
Data-Access Module
(A)
Basic access interface
Private Branch Exchange (PBX)
DMI
Primary rate-access interface
Host Computer
Integrated voice and data workstation
(B)

Figure 1—PBX
Manufacturers implementing DMI and applying the CCITT's ISDN standards will replace expensive data modules (A) with less expensive, higher-performance Basic Access and Primary Rate Interfaces (B).

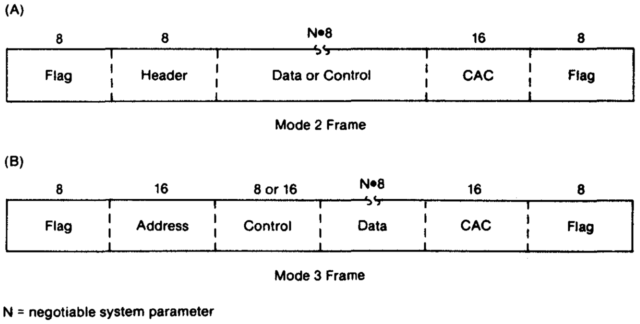


Figure 2—Service
DMI offers several protocols for transporting user data: (A), Mode 2 is a simple HDLC-based protocol optimized for supporting terminals connecting to a PBX over RS-232. (B), Mode 3 is the ISDN LAP-D protocol applied to the B-channel to support data rates approaching 64 Kb/s and statistical multiplexing.

channel T1 interface using D4 or Fe framing (DS1), common-channel bit-oriented signaling, and an HDLC packet protocol for transporting user data. The channels are 64 Kb/s; of the 24, 23 are used for carrying user data (ISDN B channels) and one is used for common-channel signaling (ISDN D channel).

In addition to specifying the interface, which provides the 23 B and 1 D channels (or 30 B and 1 D for CEPT), the DMI specification includes four formats, referred to as modes, for transporting user information. Each mode facilitates the use of the interface for a particular function. We chose to implement DMI's Mode 2—a simple HDLC protocol—because it is suitable for supporting devices that connect to the PBX network through RS-232 interfaces at speeds of 19.2 Kb/s or less (Figure 2).

Work on the interface began early in 1984 with the objective of releasing a product by late 1985. Close cooperation between HP and AT&TIS throughout development, testing, and support planning was possible because of AT&TIS commitment to DMI as an open interface. HP, as a DMI licensee and participant in the DMI Users' Group, was able to take advantage of substantial material support provided by AT&TIS in the form of integrated circuits for handling the DS1 data stream and a communication system test switch with T1 capability. The Users' Group is an independent organization of licensees that was started by AT&TIS to catalyze the development of DMI products by computer and PBX firms licensing the specification.

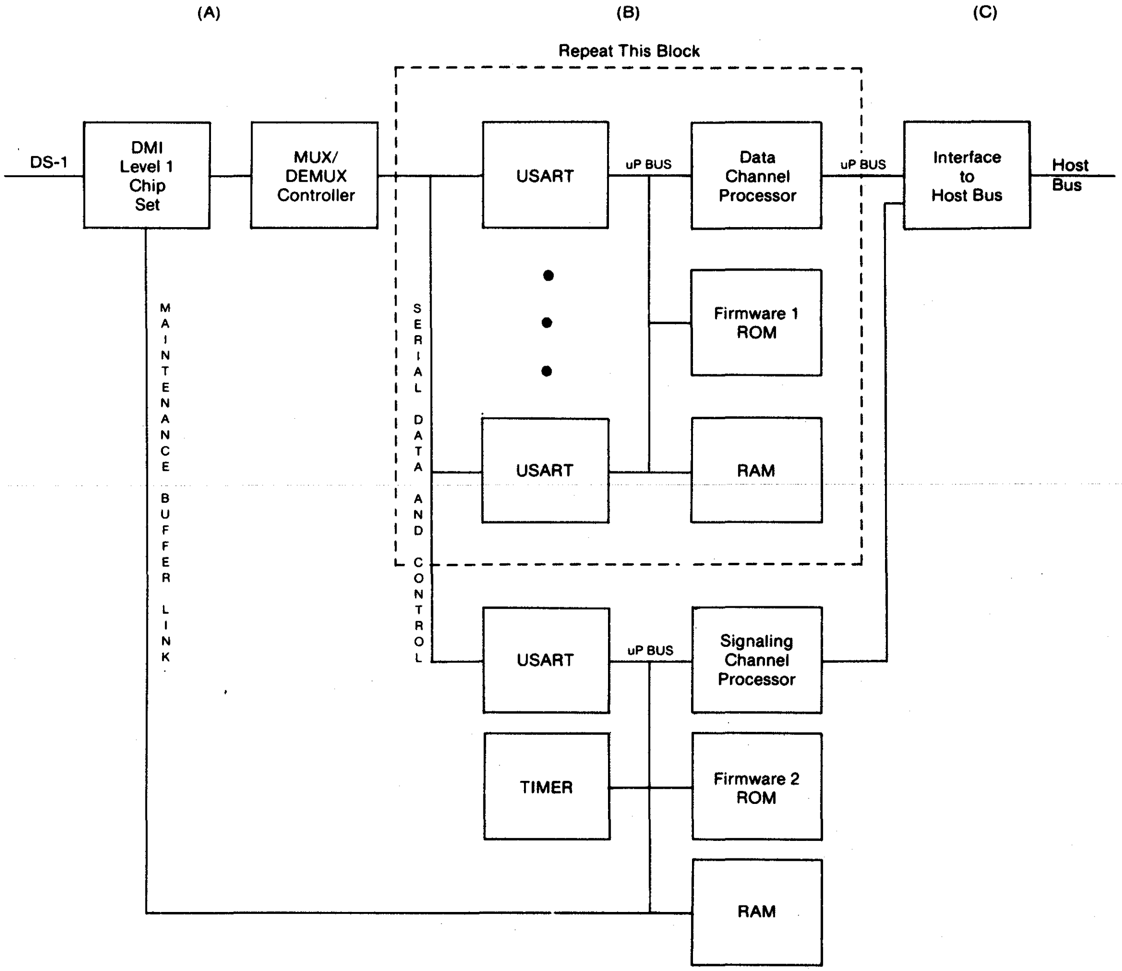


Figure 3—Overview
Three basic elements from the DMI interface: (A), the front end, which handles the T1 data stream; (B), processor modules operating on the full-duplex, 64 Kb/s channels; and (C), interface to the host processor.

Testing the interface during development and in the field was of major concern. We adopted a test strategy based on extensive built-in self-tests using a series of loopbacks that extended outward from the host to the T1 connector. This simplified the process of locating errors in the field. Development testing was done with a communication system test switch provided by AT&TIS. Once the product is completed and released it will be supported in the field by HP and AT&TIS under the terms of a certification agreement.

MAKING IT REAL

An overview of the interface (Figure 3) shows its three basic elements: the front end, which accepts and transmits the DS1 data stream and multiplexes and demultiplexes it into 24 channels; a center section, which consists of parallel processing modules for the data streams; and the interface to the host computer's backplane.

The decision about which functions to execute using firmware and which would depend on hardware was important to reaching our objective of smoothly upgrading the product to ISDN compatibility (Figure 4). Because much of the CCITT's ISDN network is based on existing standards, such as T1 and HDLC, we determined that the physical layer functions would not be changed in the ISDN standards and should be implemented using LSI–VLSI hardware available from AT&T and other vendors.

A line was drawn between hardware and firmware implementation where we got into the USARTs that would accept HDLC packets containing user data. ISDN specifies a link access protocol (LAP-D) for use in signaling on the D channel. Although we did not implement this initially, we targeted it as an objective for future upgrade. We also saw many reasons for eventually using the LAP-D protocol as the data link layer for transporting user information on the B channels (DMI Mode 3). LAP-D is similar to the LAP-B used by X.25 but contains an additional 8-bit address field in the frame's header. This difference and our desire to support a variety of protocols including DMI's Mode 3 were sufficient to warrant the use of Universal Synchronous/Asynchronous Receiver/Transmitter (USART) chips that would be flexible at the bit level. Software that runs on a microprocessor controlling the USART will take care of all of the protocol's high-layer functions.

AT&T DS1 integrated circuits provided the necessary DS1 physical layer functions in the front end of the interface. These include transmission and reception, line coding, frame recognition and generation, and synchronization. Channel multiplexing and demultiplexing were done in hardware using a design we developed (Figure 5).

Hardware Functions

T1 Interface Hardware

- DS-1 transmission/reception
- Line coding and synchronization
- Frame generation/recognition
- Channel multiplexing/demultiplexing

Data Channel Hardware

- Flag search and insertion
- Address field recognition
- CRC generation and checking
- Abort sequence generation/checking
- Interrupt capability

Firmware Functions

Data Channel Firmware

- Programming of HDLC usarts
- Control message handshaking Processing and updating
- Data transfer to and from usart
- Data metering
- Terminal character processing
- Frame sequencing, acknowledgement and retransmission
- Communication with backplane processor

Signaling Channel Firmware

- Common channel signaling
- Diagnostic of DS-1 chips
- Communication with data channel Processor

Figure 4—Implementation

We chose a Zilog Z80 linked to RAM, ROM, and a Zilog USART to implement the modules we used for handling multiple data streams from the DS1 front end. Low risk, flexibility, and design experience were the dominant factors leading to this choice. A single-state machine for the Mode 2 protocol was stored in memory and used by the microprocessor to run the protocol on multiple channels. This timesharing arrangement reduced the requirement for memory because the program size in memory was considerably larger than the memory necessary to store the context variables for each of the individual channels.

DMI's Mode 2 protocol was our choice for initial implementation because it is the most suitable for supporting terminals that attach to the PBX (usually through a data module) using an RS-232 interface. By using a cyclic redundancy checksum (CRC) as part of each frame, Mode 2 provides end-to-end error detection for user information as it passes through the PBX from a data module to the host. RS-232 lead information is transported by Mode 2 in special control frames, allowing the use of dial-in modems at the terminal interface.

The Z80 microprocessor module design used for the B channels also was used on the D channel. For signaling, a simple bit-oriented protocol based on the use of A and B bits was implemented. Unlike the traditional robbed-bit signaling, which periodically writes signaling information for a given channel onto the channel's data stream (requiring the use of a special format that restricts 64-Kb/s channels to 56 Kb/s), all signaling across the DMI interface is restricted to the common D channel.

UPGRADING THE INTERFACE: LEVERAGING A MODULAR DESIGN

We have focused on the implementation of our initial DMI product, but it is important to consider the enhanced capabilities provided by upgrading the product via firmware and the effect of VLSI device support. Many of the benefits of ISDN for data communications customers depend on a standard, strict interpretation fo the ISDN standards and on the ability of manufacturers to build onto the facilities provided in an ISDN interface. Common-channel message-oriented signaling as specified by the ISDN standards is a good example. With this dramatic improvement over the bit-robbed signaling that has been used traditionally, a new basis for improved communication between network resources is provided. Improved communication in turn provides a facility for better network management (maintenance and security, for example). This capability will be added to our existing product through firmware upgrade.

With the advent of inexpensive standard ISDN basic access interfaces for terminal products (Advanced Micro Devices has announced that integrated circuits implementing this interface will be available in 1985) there will be an increased need to handle user data rates that approach 64 Kb/s. This need can be met by changing the protocol supported on our product from DMI Mode 2, which is limited to RS-232 data rates, to

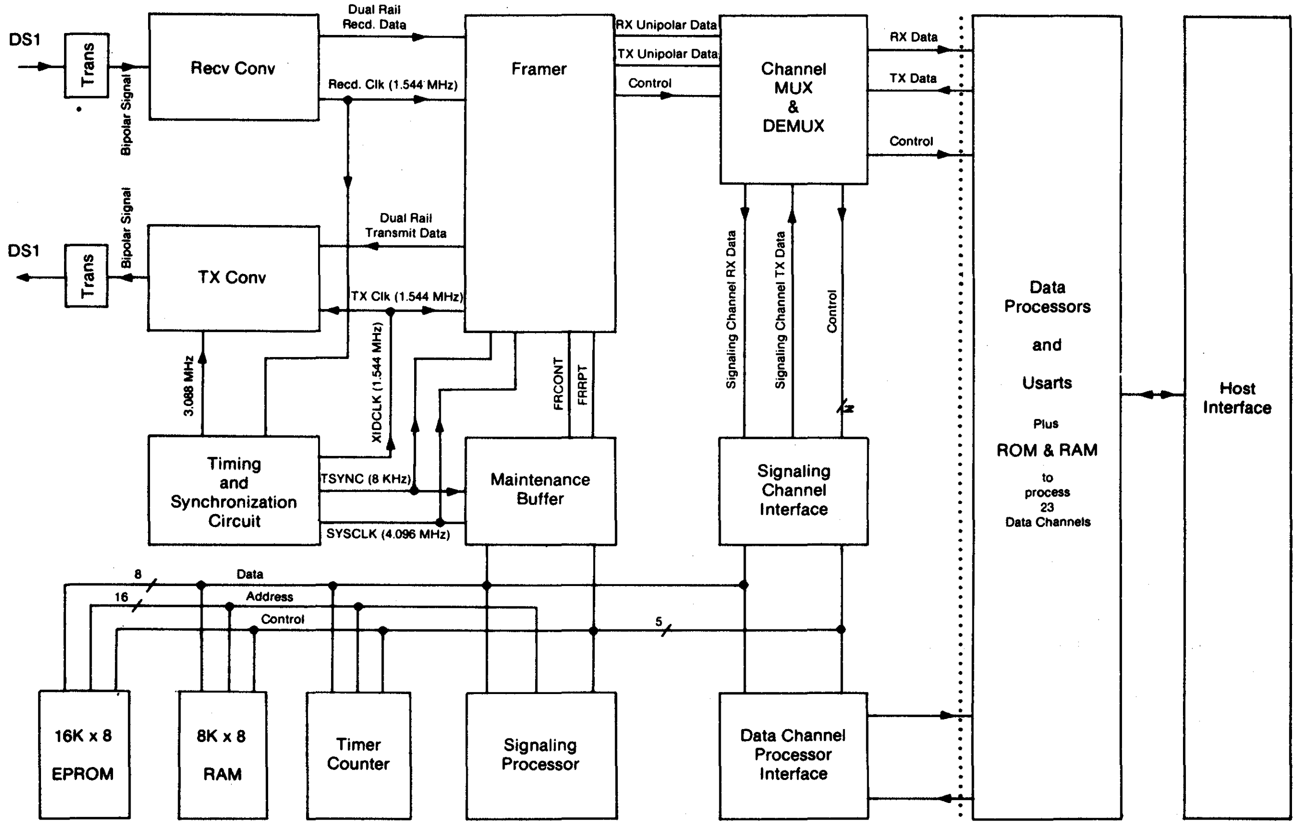


Figure 5—Front end

DMI Mode 3, which uses the LAP-D protocol. One of several important advantages of LAP-D is its ability to allow several different data links to exist on a single, circuit-switched connection. Applying this to a DMI interface allows the use of a packet switch either on the terminal side of the PBX or within the PBX itself to concentrate the traffic from multiple terminal devices onto a single DMI B channel. If the terminal devices have an RS-232 connection to the packet switch (or the PBX, if it contains the packet switch) then several low-speed terminals can be accommodated within a single 64-Kb/s B channel. This will provide substantial increase in the use of the interface.

VLSI device support for DMI (Rockwell and other integrated circuit vendors have announced products for 1985) is focused on integrating the processing elements used to handle the simultaneous streams of HDLC packets on different B channels. In addition to performing normal HDLC functions, these devices will have built-in DMA capability. A substantial decrease in the board space required will be possible using these devices instead of the microprocessor module architecture we have implemented using off-the-shelf integrated circuits.

ECONOMICAL OFFICE NETWORKING

HP's decision to implement DMI resulted from our awareness of the increasing importance of the PBX in office automation. We also considered the importance of ISDN in providing a standard solution that would be supported well by the PBX and computer industries. The DMI specification provided an open-interface architecture that we could develop immediately and that later could migrate smoothly and economically to ISDN compatibility while providing substantial cost advantages over existing products.

Close cooperation between HP and AT&TIS through the DMI User's Group was instrumental in allowing the successful development of HP's DMI product. This cooperation stemmed from the commitment both companies have made to ISDN and in particular from the AT&TIS commitment to DMI as an open interface. Design aids and information on device support were readily accessible through the DMI User's Group.

Immediate implementation of the DMI and the ability to upgrade to ISDN compatibility via firmware were important objectives that were balanced against the need to make the product inexpensive enough to provide substantial cost advantages over existing PBX-based solutions. Achieving these objectives was possible because good approaches to design were taken using existing integrated circuits. Strong support from AT&TIS also was important.

Our initial DMI product provides a stepping-stone to further ISDN interface development and more economical office networking. The commitment of manufacturers to work together will continue to grow in importance for customers, who increasingly value compatibility among vendors, and for manufacturers, who will reap the benefits of increased market size.

Panel: Innovation and industry standards—networking issues

Chair:
THOMAS H. BREDT, *Dataquest Incorporated*, San Jose, California
Members:
SUBASH BAL, *Excelan,* San Jose, California
RAYMOND DONNELLY, *AT&T Information Systems,* Morristown, New Jersey
MARK HAHN, *Corvus Systems, Inc.,* San Jose, California
PHIL WHITE, SR., *Altos Computer Systems,* San Jose, California

Innovation and standards: Is there room for both in the local area network market? This panel brings together leaders in local area network technology, who discuss the most pressing issues they face today. Panelists address alternatives to networking, the future of some of the more popular LANs, and the race to endorse a standard solution. Other issues are the economic aspects of networks: connection costs, and buying technology versus designing it in house.

Panel: The arrival of LAN operating systems

Chair:
NOEL E. SCHMIDT, *Architecture Technology Corporation,* Minneapolis, Minnesota
Members:
CRAIG BURTON, *Novell, Inc.,* Orem, Utah
MICHAEL LOFTUS, *Digital Research,* Monterey, California
GERALD POPEK, *Locus Computing Corp.,* Santa Monica, California
JOHN ROW, *Applied Intelligence,* Mountain View, California

By providing general-purpose resource sharing and communications services to application developers, LAN operating systems are facilitating user acceptance of LAN products. Details of topology, protocols, node addressing, and heterogeneous devices are effectively masked from the user. This session's panelists speak to issues relevant to selecting LAN operating systems, including security, logical naming, access privileges, and network communications. They also address characteristics of products available in the marketplace.

Panel: Fiber optics in local area networks

Chair:
JAMES Y. BRYCE, *Netserv, Inc.,* Austin, Texas
Members:
MICHAEL CODEN, *Codenoll Technology,* Yonkers, New York
JAY CUNNINGHAM, *Siecor Electro-Optical Products,* Research Triangle Park, North Carolina
HOWARD SALWEN, *Proteon Inc.,* Natick, Massachusetts
MICHAEL SEE, *IBM,* Research Triangle Park, North Carolina
FRED WEST, *Hewlett-Packard,* Richardson, Texas

The benefits of fiber optics—wide bandwidth, freedom from EMI, low error rate, safety, and absence of grounding problems, among others—are still not fully appreciated. Telephone communication is rapidly adopting fiber; local area networks will soon follow. Greater flexibility in topology and the implementation of existing networks on such de facto standards as the IBM cabling system will be heightened by introduction of fiber optics. This introduction will occur more rapidly than many people now realize. The panel explores the possibilities of fibers, markets, and products available for use in local area networks.

Panel: Networks in the factory

Chair:
DANIEL E. GAHLON, *Interactive Systems/3M,* St. Paul, Minnesota
Members:
GARY HENKEL, *Harris Corporation,* Melbourne, Florida
RON KEIL, *GMC GM Technical Center,* Warren, Michigan
STEVE LaPORTE, *Medtronic, Inc.,* Fridley, Minnesota

This panel discussion features the experiences of several users of large broadband local area networks. The presentations focus on current applications and future requirements. Included in the discussions are information on standards requirements in hybrid networks. Of special interest in the presentations is discussion of manufacturing applications, with a particular slant toward the IEEE 802.4 and MAP specifications.

Panel: PC communications systems

Chair:
JUDITH ESTRIN, *Bridge Communications,* Mountain View, California
Members:
BARBARA KOALKIN, *Apple Computer,* Cupertino, California
BOB METCALFE, *3 Com,* Mountain View, California
ROBERT J. ZACK, *Digital Communications Associates,* Norcross, Georgia

Personal computers now are being used in a variety of environments: stand-alone applications in the home and office, workgroup applications in small businesses and corporations, and workstation applications as part of a large corporate network. The speakers on this panel present their views on the requirements of these different applications, as well as the products that their companies market to address these requirements.

Panel: Impact of the IBM PC network

Chair:
GREGORY ENNIS, *Sytek Corp.,* Mountain View, California
Members:
DAVE MELIN, *Microsoft,* Bellevue, Washington
WILLIAM OSBORNE, *IBM-ESD,* Delray Beach, Florida
SIGNE OSTBY, *Software Publishing,* Mountain View, California

Does the IBM PC network, with its rich set of network services, mark a new opportunity for applications and application writers? Panelists discuss their perspectives on the impact of the IBM PC network and PC networking in general and their visions of where PC applications are headed.

Panel: Future of networking

Chair:
NORMAN F. SCHNEIDEWIND, *Naval Postgraduate School,* Monterey, California
Members:
GARY S. CHRISTENSEN, *Network Systems Corporation,* Minneapolis, Minnesota
PHIL EDHOLM, *Sytek,* Mountain View, California
DENNIS KING, *Tandem Computers, Inc.,* Cupertino, California

The future of computer networking is examined by executive and product planners from three vendors. The panelists project future products and services in the areas of supernetworks, network software, user area wiring and workstations, bridges and gateways between networks, intelligent networks and user productivity.

1985 NATIONAL COMPUTER CONFERENCE COMMITTEES

PROGRAM COMMITTEE

Chairman
Anthony S. Wojcik
Illinois Institute of Technology
Chicago, IL

Administrative Assistant
Matthew Bauer
Illinois Institute of Technology
Chicago, IL

Proceedings Coordinator
Anneliese von Mayrhauser
Illinois Institute of Technology
Chicago, IL

Program Vice-Chairmen
C. Robert Carlson
Illinois Institute of Technology
Chicago, IL

Marvin Ehlers
Natural Gas Pipeline of America
Lombard, IL

Harvey Freeman
Architecture Technology Corporation
Minnetonka, MN

James Gerdes
Argonne National Laboratory
Argonne, IL

Samir Husson
International Business Machines, Inc.
Poughkeepsie, NY

Kai Hwang
University of Southern California
Los Angeles, CA

Ewing L. Lusk
Argonne National Laboratory
Argonne, IL

David Rine
George Mason University
Fairfax, VA

Program Track Assistants
Victor Janulaitis
Positive Support Review, Inc.
Los Angeles, CA

Robert Kuhn
Gould, Inc.
Rolling Meadows, IL

Barbara McNurlin
Canning Publications
Torrance, CA

CONFERENCE STEERING COMMITTEE

General Chairman
Karl E. Martersteck
AT&T Bell Laboratories
Naperville, IL

Vice-Chairman
Nicholas L. Marselos
AT&T Network Systems
Lisle, IL

Program Chairman
Anthony S. Wojcik
Illinois Institute of Technology
Chicago, IL

Program Vice-Chairman
C. Robert Carlson
Illinois Institute of Technology
Chicago, IL

Secretary
David H. Jacobsohn
Argonne National Laboratory
Argonne, IL

Financial Chair
Barbara H. Van Husen
Technical Support Group, Inc.
Chicago, IL

Implementation Logistics Chairman
William J. Carlisle
Bell Communications Research, Inc.
Piscataway, NJ

Operations Chairman
Richard B. Wise
Sargent & Lundy Engineers
Chicago, IL

Pioneer Day Chair
Margaret K. Butler
Argonne National Laboratory
Argonne, IL

Professional Development Seminars Chairman
Robert R. Gershon
Gershon Associates
Deerfield, IL

Promotion Chairman
Donald G. Dowd
Don Dowd Communications, Inc.
Belmont, CA

Special Activities Chair
Mary W. Owen
SPSS, Inc.
Chicago, IL

NCCC Liaison
Albert K. Hawkes
Sargent & Lundy Engineers
Chicago, IL

COMMUNICATIONS AND PROMOTION COMMITTEE

Chairman
Donald G. Dowd
Don Dowd Communications, Inc.
Belmont, CA

Members
Sally Berger
Amherst Associates, Inc.
Chicago, IL

Matthew Carlson
News and Communication Services, Ltd.
Chicago, IL

Daniel Collins
Bell and Howell Company
Skokie, IL

Susan Croft
Susan Croft and Associates
Los Angeles, CA

Victor J. Danilov
Museum of Science and Industry
Chicago, IL

James Eaton
Ericsson, Inc.
Garden Grove, CA

Mardi Garren
Ramtek Corporation
Santa Clara, CA

Sherry Goodman
Chicago Access Corporation
Chicago, IL

Roger Halligan
Halligan & Associates
Chicago, IL

Evelyn Heaton
Illinois Software Association
Chicago, IL

Evelyn Hoffman
Illinois Department of Commerce
Chicago, IL

Peter N. Holste
Information Systems/3M
St. Paul, MN

Barbara Krause
Apple Computer, Inc.
Cupertino, CA

Rita Lane
Blue Cross/Blue Shield
New York, NY

William LeMaster
NCR Corporation
Dayton, OH

Janet Lerman
Burroughs Corporation
Detroit, MI

E. Z. Million
Consultant
Norman, OK

Ross L. Nelson
Miller Smith, Inc.
Denver, CO

Kent R. Nichols
Control Data Corporation
Minneapolis, MN

Norman Peterson
Governor's Commission on Science and Technology
Chicago, IL

Thomas R. Ray
Keycom Electronic Publishing
Schaumburg, IL

Patricia S. Restaino
Votan
Fremont, CA

Herbert B. Safford
GTE Data Services, Inc.
Marina Del Rey, CA

Steven Sester
Ruder, Finn and Rotman
Chicago, IL

William D. Stotesbery
Microelectronics & Computer Technology Corporation
Austin, TX

Donald Tanguay
Sperry Information Systems
Mississauga, Ontario, Canada

Marlene Williamson
Zenith Data Systems
Glenview, IL

M. Mildred Wyatt
Wyatt Communications, Inc.
Chicago, IL

OPERATIONS COMMITTEE

Chairman
Richard B. Wise
Sargent & Lundy Engineers
Chicago, IL

Members
Maribeth Andrews
First National Bank
Chicago, IL

Shirley A. Baird
Milestone Systems, Ltd.
Downers Grove, IL

Earl J. Calkins
Sargent & Lundy Engineers
Chicago, IL

Vince Cunningham
Illinois Bell
Chicago, IL

Kenneth Disch
American Medical Association
Chicago, IL

Leslie A. Eaton
AT&T Computer Systems Center
Lisle, IL

Joseph A. Leubitz
Accelerated Computer Training
Northbrook, IL

Michael Weiland
UOP Process Division
Des Plaines, IL

PIONEER DAY COMMITTEE

Chair
Margaret K. Butler
Argonne National Laboratory
Argonne, IL

Members
James H. Alexander
S-Cubed
San Diego, CA

Earl W. Burdette
U.S. Department of Energy
Oak Ridge, TN

Harry W. Conner
Argonne National Laboratory
Argonne, IL

Miriam L. Holden
Argonne National Laboratory
Argonne, IL

Alston S. Householder
Malibu, CA

Edwin L. Hughes
Melbourne, FL

David H. Jacobsohn
Argonne National Laboratory
Argonne, IL

Paul Messina
Argonne National Laboratory
Argonne, IL

James E. Robertson
University of Illinois—Urbana
Urbana, IL

Alfred J. Rucker
Argonne National Laboratory
Argonne, IL

PROFESSIONAL DEVELOPMENT SEMINARS COMMITTEE

Chairman
Robert R. Gershon
Gershon & Associates
Deerfield, IL

Members
Charles Balsly
Perle GSD
Chicago, IL

Selwyn Becker
University of Chicago
Chicago, IL

Joseph Gershon
Educational Consultant
Mt. Prospect, IL

Edward Topor
Edward S. Topor and Associates
Chicago, IL

SPECIAL ACTIVITIES COMMITTEE

Chair
Mary W. Owen
SPSS, Inc.
Chicago, IL

Members
Marjorie Benson
University of Chicago
Chicago, IL

Jane Kantowicz
CNA Insurance
Chicago, IL

Matthew Kline
Kline and Co.
Chicago, IL

Harvey Marks
TTI, Inc.
Santa Monica, CA

Joseph McGarry
SRA/IBM
Chicago, IL

Keiichi Sato
Illinois Institute of Technology
Chicago, IL

Linda Vermillion
The Kroger Co.
Cincinnati, OH

Marylouise White-Petteruti
Leo Burnett Co.
Chicago, IL

NCC '85 SESSION LEADERS

Dharma P. Agrawal
North Carolina State University
Raleigh, NC

James H. Alexander
S-Cubed
San Diego, CA

Adarsh K. Arora
Gould, Inc.
Rolling Meadows, IL

Rick Baker
Hewlett-Packard Co.
Corvallis, OR

Mario Barbacci
Carnegie-Mellon University
Pittsburgh, PA

D'Ellen Bardes
Alltech Communications
Denver, CO

Eamon Barrett
Smart Systems Technology
McLean, VA

John L. Berg
Sperry Corporation
St. Paul, MN

Hal Berghel
University of Nebraska
Lincoln, NE

Robert W. Blanning
Vanderbilt University
Nashville, TN

William E. Bracker, Jr.
University of Arizona
Tucson, AZ

William Bramer
Arthur Andersen & Co.
Los Angeles, CA

Thomas H. Bredt
Dataquest Incorporated
San Jose, CA

Fayé A. Briggs
Rice University
Houston, TX

David R. Brodwin
Arthur D. Little, Inc.
San Francisco, CA

Steven Brower
Wood, Lucksinger & Epstein
Los Angeles, CA

James Y. Bryce
Netserv, Inc.
Austin, TX

C. R. Carlson
Illinois Institute of Technology
Chicago, IL

Richard P. Case
IBM Corporation
White Plains, NY

Edward Cawi
Elgin School District
Elgin, IL

S. K. Chang
Illinois Institute of Technology
Chicago, IL

Sylvia Charp
Past President, AFIPS
Upper Darby, PA

J. Chuan Chu
Santec Corporation
Amherst, NH

Thomas B. Cross
Cross Information Co.
Boulder, CO

Peter Dadam
IBM Corporation
Heidelberg, West Germany

Henry Delevati
Information Builders, Inc.
San Jose, CA

Stephen J. Durham
Cermetek Microlectronics, Inc.
Sunnyvale, CA

Gregory Ennis
Sytek Corporation
Mountain View, CA

Raymond Epich
Northwest Industries
Chicago, IL

Judith Estrin
Bridge Communications
Mountain View, CA

Martha W. Evens
Illinois Institute of Technology
Chicago, Illinois

Kenneth Forbus
University of Illinois—Urbana
Urbana, IL

Daniel E. Gahlon
Interactive Systems/3M
St. Paul, MN

Daniel Gajski
University of Illinois—Urbana
Urbana, IL

Jean L. Gaudiot
University of Southern California
Los Angeles, CA

Steven Gilman
Information Builders, Inc.
El Segundo, CA

Dennis R. Goldenson
Carnegie-Mellon University
Pittsburgh, PA

Randy J. Goldfield
The Omni Group
New York, NY

William C. Goss
International Bureau of Software Test
Sunnyvale, CA

Paul L. Hazan
Johns Hopkins University
Laurel, MD

Carl T. Helmers, Jr.
North American Technology, Inc.
Peterborough, NH

Alex A. Hoffman
Consultant
Fort Worth, TX

Christine G. Hughes
Gartner Group, Inc.
Stamford, CT

Patricia L. Hutinger
Western Illinois University
Macomb, IL

Kai Hwang
University of Southern California
Los Angeles, CA

Armond Inselberg
Synapse Computer Corporation
Milpitas, CA

M. Victor Janulaitis
Positive Support Review, Inc.
Los Angeles, CA

Laurel V. Kaleda
IBM Corporation
San Jose, CA

Arnold E. Keller
Hitchcock Publishing Co.
Wheaton, IL

Lawrence R. Kilty
The Kilty Company
Bethesda, MD

Dale F. Lake
Wickes Companies, Inc.
Santa Monica, CA

Richard L. Linting
Arthur Andersen & Co.
Chicago, IL

Howard J. MacKenzie
Stamco Office Products
Chicago, IL

Gerald Q. Maguire, Jr.
Columbia University
New York, NY

Jackie Marsh
Motorola, Inc.
Austin, TX

Richard Mateosian
National Semiconductor Corporation
Berkeley, CA

Kathleen R. McKeown
Columbia University
New York, NY

Ephraim McLean
UCLA Graduate School of
Management
Los Angeles, CA

Alan Paller
AUI Data Graphics/ISSCO
Arlington, VA

George Paul
IBM Corporation
Yorktown Heights, NY

John Payne
National Semiconductor Corporation
Santa Clara, CA

Michele K. Pesta
AT&T Information Systems
Summit, NJ

Walter Popper
Index Systems
Cambridge, MA

William G. Rankin
Deere & Co.
Moline, IL

John R. Rice
Purdue University
West Lafayette, IN

George Rittersbach
Peat, Marwick, Mitchell & Co.
Chicago, IL

Terry R. Savage
TRW
Redondo Beach, CA

Noel E. Schmidt
Architecture Technology Corporation
Minneapolis, MN

Norman F. Schneidewind
Naval Postgraduate School
Monterey, CA

David L. Shay
N. Dean Meyer & Associates, Inc.
Ridgefield, CT

Howard J. Siegel
Purdue University
West Lafayette, IN

John P. Singleton
Security Pacific Automation Co.
Los Angeles, CA

James Slagle
University of Minnesota
Minneapolis, MN

Alan Sobel
Lucitron, Inc.
Northbrook, IL

David Sprague
Western Illinois University
Macomb, IL

Duvurru Sriram
Carnegie-Mellon University
Pittsburgh, PA

Edward H. Sussenguth
IBM Corporation
Research Triangle Park, NC

James Vandendorpe
AT&T Bell Laboratories
Naperville, IL

Pramode K. Verma
AT&T Information Systems
Lincroft, NJ

Anneliese von Mayrhauser
Illinois Institute of Technology
Chicago, IL

Marilyn D. Ward
Western Illinois University
Macomb, IL

Amy D. Wohl
Wohl & Associates
Bala Cynwyd, PA

Lawrence Wos
Argonne National Laboratory
Argonne, IL

Nicholas Zvegintzov
Editor, *Software Maintenance News*
Staten Island, NY

NCC '85 REFEREES

Adams, Chuck
Agrawal, Dharma P.
Albert, Theodore M.
Albin, Marvin
Allison, Dennis
Amamiya, Makoto
Archibald, Julius A., Jr.
Aurbach, Richard L.

Ballard, Bruce W.
Ballew, Lowell N.
Batcher, Kenneth
Bauman, Ben M.
Beidler, John
Belford, Geneva
Ben-Menachem, Mordechai
Bhat, M. V.
Bhavsar, V. C.
Bocast, Alexander
Bogner, Sue
Borko, Harold
Braun, Chris
Brocato, Lou
Brodwin, David R.
Brower, Steven
Brown, Richard G.
Bui, Tung
Burden, Richard L.

Canas, Daniel A.
Cannon, George R., Jr.
Carney, Homer C.
Casaglia, Gianfranco
Celko, Joe
Chapin, Ned
Chen, Chi H.
Chuang, Henry Y. H.
Chung, Won L.
Cook, Curtis R.
Cormier, Robert W.
Cross, F.
Cross, Thomas B.
Cunningham, R. S.

Dankel, Douglas D., II
Davis, James
Day, William H. E.
Dearholt, Don
Deogun, Jitender S.
Dordick, H. S.
Dourandish, Robert
Dyer, Michael
Dyment, Doug

Eliot, Lance B.
Eller, Thomas J.
Elmaghraby, Adel S.
Ernst, Dennis
Esbin, Jack
Evens, Martha
Evert, Carl F.

Fendrich, John
Fernandez, Eduardo B.
Field, George
Finn, Nancy B.
Fischer, Diane
Fischer, Herman
Flowers, Margot
Frailey, Dennis
Frederick, Terry J.
Friedman, Frank

Gabrielsen, Larry L.
Gaines, Brian R.
Gannon, Steve
Garrett, Leonard
Geist, Harold
Gessford, John F.
Gintz, Christopher
Glaseman, Steven
Goel, Amrit L.
Goldberg, Jack
Gorsline, G. W.
Gottlieb, Allan
Grosky, William
Grossman, Harold C.
Guerrieri, Ernesto

Hac, Anna
Hadavi, Khosrow
Harris, Michael C.
Heafner, John
Heath, Douglas A.
Heblinger, George H.
Higginbotham, T. F.
Hill, Thomas L.
Hintz, Joseph C.
Hirst, Graeme
Horch, John
Huhns, Michael N.
Hurson, Ali R.

Irby, Thomas C.

Jacobs, Steven M.
Jackson, Durward P.
Jagannathan, V.
Jain, Hemant K.
Jehn, Lawrence A.
Johnson, James J.
Johnson, James Lee
Jones, Douglas W.
Juell, Paul
Jakobson, Gabriel E.
Kamel, Khaled
Kant, K.
Kearns, Phil
Kenett, Ron
Kessler, R. A.
King, Roger
Kira, Nicholas J., III
Kirby, Peter G.
Kluczny, Raymond M.
Kolstad, Rob
Kumar, Vipin
Kundu, S.

Lake, Robin B.
Langdon, David M.
Larson, Arvid G.
Lawrence, Robert
Lee, Daniel T.
Leinius, Ronald P.
Lillevik, Sigurd L.
Linn, Joseph L.
Little, Joyce
Little, Rainey
Liu, Hsun K.
Lively, William
Loach, K. W.
Loomis, Mary E.

MacAuslan, Joel
MacEwen, G. H.
Magel, Kenneth
Mamone, Salvatore
Marriott, Philip C.
Mashaw, Bijan
Mateosian, Richard
Mauney, J.
McClure, Carma
McCullough, Tim
McKee, James
McMillen, Robert J.
Medley, Don B.
Messner, Stephen C.
Metzner, John R.
Milutinovic, Velco
Mitchell, Don B.
Mok, Al K.
Molloy, Michael K.
Motzkin, Dalia
Mudge, Trevor
Mullens, David G.
Murray, Dale N.

Naifeh, Kim L.
Navlakha, Jainendra
Neblock, Charles E.
Newman, Edward
Neilson, Gregory M.

Okawa, Yoshikuni
Olavesen, Ole Bernt

Palmer, Harlan
Patt, Yale
Perkins, Sharon
Peterson, Emery G.
Peterson, Robert W.
Potter, Jerry L.
Potter, Marshall R.
Pramanik, Sakti
Pyron, Howard D.

Rault, J. C.
Reed, Daniel A.
Riganati, John P.
Rine, David C.
Robb, Richard A.
Roberts, Philip
Rollwitz, Bill
Romanowsky, Helen E.
Rosen, Robert
Rosenberg, Steven
Rosenthal, Paul H.
Roth, R. Waldo
Ruh, Lawrence A.

Schaefer, Hans
Schell, Roger R.
Schindler, Max
Schmookler, Martin
Schrader, David K.
Schueppert, Robert G.
Schutzer, Daniel
Schwartz, David P.
Sekino, Akira
Serlin, Omri
Sharp, Donald D., Jr.
Shaughnessy, Edward P.
Shaw, Mildred L. D.
Shetler, Toni
Silver, Aaron N.
Simmons, Dick B.
Smartt, Melissa
Smith, J. W.
Smith, Lyle B.
Smith, Richard A.
Smith, S. Diane
Smoliar, Stephen W.
Soffa, Mary Lou
Soloway, Elliot
Spaniol, Roland
Spresser, Diane M.
Stavely, Allan M.
Stern, Richard H.
Stockman, George
Sutter, Gordon F.
Swigger, Kathleen M.

Tanik, Murat M.
Tanniru, Mohan R.
Taylor, Linda T.
Taylor, Javin
Tharp, Alan L.
Thebaut, S. M.
Trauth, Eileen M.
Treu, Siegfried

Uckan, Yuksel
Umbaugh, L. David

Wallace, Dolores R.
Wang, C. Charles
Wang, R. S.
Ware, Joel
Waxelblat, R. L.
Whitesell, James T.
Weigler, Barry
Williams, Donald S.
Wilson, Gerald A.
Woodfill, Marvin C.
Wortz, Charles
Wu, Chuan-lin

Yost, Robert
Young, Glenn

Zhou, Zhi Ying
Zoll, Peter F.
Zunde, Pranas

NCC '85 SPEAKERS AND PANELISTS

Adams, Russ
Agrawal, Dharma P.
Albert, M. Scott
Alles, Hal
Andrews, Robert
Appelt, Douglas E.
Arnold, Robert
Arora, Adarsh K.

Baker, Rick
Bal, Subash
Balcom, L. B.
Ballhaus, William F.
Barbacci, Mario
Bardes, D'Ellen
Barnard, H. Jack
Barrett, Eamon
Batz, Joseph
Beck, Ken
Belady, Les
Bell, C. Gordon
Benne, Bart
Berg, John L.
Berghel, Hal
Bic, Lubomir
Blanning, Robert W.
Blaylock, Johelen
Bledsoe, Woody
Blum, Michael
Borkin, Sheldon A.
Botts, Doug
Brachman, Fred
Brackbill, Doug
Bracker, William E., Jr.
Bramer, William
Bredt, Thomas H.
Brice, Linda
Bride, Edward J.
Briggs, Fayé A.
Bright, Roy
Brodwin, David R.
Brooks, Rubin
Brower, Steven
Brown, Geoffrey
Brown, Jonathan A.
Bryce, James Y.
Buchmann, Alex
Burkhart, Roger M.
Burns, Nina
Burton, Craig
Butler, James W.
Bylander, Thomas

Campbell, J. Olin
Cargo, David
Carlson, C. R.
Cashen, Joseph
Casto, Randy
Cawi, Edward
Chandhok, Ravinder P.
Chandrasekaran, B.
Chang, S. K.
Chapin, Ned
Charp, Sylvia
Christensen, Gary S.
Coden, Michael
Collofello, James S.
Connell, John L.
Connolly, John W. D.
Couger, J. Daniel
Cramp, John
Cross, Thomas B.
Crowell, Peter
Cunningham, Jay
Curtis, Walt
Czaplicki, David

Dadam, Peter
Davida, George I.
Davis, Nathaniel J., IV
deKleer, Johan
Delevati, Henry
DeWitt, Norman R.
Dickson, Gary W.
DiMarzio, Alfred W.
Dittrich, Klaus
Dodd, George
Donnelly, Raymond
Duffy, Frank
Downing, Arthur C.
Durham, Stephen J.
Dykeman, John B.

Edholm, Phil
Eidenberg, Eugene
Eldon, John
Ennis, Gregory
Epich, Raymond
Epstein, Sam
Erdei, Michael
Erickson, Wayne J.
Ernst, Ron
Estrin, Judith
Evens, Martha

Fairbairn, Doug
Fang, Kwang-Ya
Farrag, Abdel A.
Fenves, Steven J.
Flies, Bill
Forbus, Ken
Fortelka, Robert
Franklin, Stephen
Frederick, Earl J.
Furukawa, Zengo

Gahlon, Dan E.
Gajski, Daniel
Gardy, Martin
Garlan, David
Gaudiot, Jean L.
Gear, C. W.
Gilman, Steve
Godden, Kurt
Goldenson, Dennis R.
Goldfield, Randy J.
Gooby, Matt
Goss, William C.
Grafton, Robert B.
Graham, Robert L.
Greenfield, Arnold
Greeson, Owen
Gross, Victor C.

Haberman, Nico
Hahn, Mark
Hall, Jean F.
Hall, W. Eric
Hambrusch, Susanne E.
Hansen, Don
Harrigan, John
Harshbarger, Katherine Y.
Harwood, Marcus
Hazan, Paul L.
Hefner, John
Helmers, Carl T., Jr.
Henderson, John C.
Henkel, Gary
Henschen, Larry
Herron, Steven D.
Herschel, Michael S.
Hershey, Ernest A.
Hinchman, John
Hirsch, Peter M.
Hoff, David
Hoffman, A. A. J.
Horch, John
Hughes, Christine G.
Hughes, Edwin L.
Hutinger, Patricia L.
Hwang, Kai

Inselberg, Armond D.

Janulaitis, M. Victor
Jenkins, Milton
Jenkins, Russel W.
Johnson, Randy
Johnsson, S. Lennart
Jones, H. F.

Jones, Helayne
Jones, Henry W., III

Kaleda, Laurel V.
Kaplan, S. Jerrold
Keil, Ron
Keller, Arnold E.
Kennedy, Miles H.
Kenny, Doug
Kilty, Lawrence R.
Kim, Jin H.
King, Dennis
Kirk, Robert S.
Kislowski, Richard J.
Kitchener, Robert
Klein, Rudolph J.
Kleinman, Neil
Koalkin, Barbara
Konsynski, Benn R., III
Korfhage, Margaret A.
Kotera, Hiroaki
Kotlowski, Conrad J.
Kroin, Alfred S.
Kuhn, Robert
Kutnick, Dale

Lake, Dale F.
LaPorte, Steve
Lerner, Mark D.
Linebarger, Robert
Link, John T.
Linting, Richard
Lipovski, G. J.
Loftus, Michael
Lorig, Gordon
Lovely, Howard
Lowenthal, Eugene
Lu, P. M.
Luetke-Stahlman, B.
Luetke-Stahlman, Kent

Mace, Rick
Machover, Carl
MacKenzie, Howard J.
Magnuson, Leonard
Maguire, Gerald Q., Jr.
Manchester, James R.
Marriott, Philip C.
Marselos, Nicholas
Marsh, A. Jackie
Martin, Nancy
Matelan, Nicholas
Mateosian, Richard
McAllister, William
McClure, Carma L.
McDonald, David D.
McKeown, Kathleen R.
McLean, Ephraim
Meador, C. Lawrence

Melin, Dave
Metcalfe, Bob
Meyer, N. Dean
Miles, Gary
Mohammed, John
Morrison, Morgan
Morton, Clifford A.
Moses, Kurt
Motzkin, Dalia
Mudge, Trevor
Murray, W. MacDonald
Myers, Robert A.

Navlakha, Jainendra
Neigh, James L.
Nemovicher, C. Kerry
Ness, David
Ng, Benjamin
Nolen, James S.

O'Connor, Dennis
Osborne, William
Otsby, Signe

Paller, Alan
Panyan, Marion C.
Parry, Arthur E.
Patel, Anil
Patterson, John
Patton, Peter C.
Paul, George
Payne, John
Pesta, Michele K.
Peterson, Victor L.
Phillips, Susan H.
Popek, Gerald
Popper, Walter

Qadah, Ghassan Z.

Rajagopalan, Raman
Rankin, William G.
Rappaport, Wanda
Rattner, Justin
Reddy, Raj
Rhodes, Wayne L.
Rice, John R.
Rittersbach, George
Roche, Timothy P.
Rosenow, Peter
Rothchild, Edward S.
Row, John
Rudin, Harry

Salwen, Howard
Savage, Terry R.
Savit, Carl
Schmidt, Noel E.
Schneidewind, Norman F.

Schorr, Herbert
Schutzer, Dan
Scott, David C.
See, Michael
Serrano, Clara I.
Shafer, Timothy C.
Shaw, Mildred L. G.
Shay, David L.
Siegel, Howard Jay
Simmons, Reid
Singleton, John P.
Slagle, James
Smarr, Larry
Smith, Doug
Smith, Reid
Sobel, Alan
Souder, H. Ray
Sprague, David
Sprague, Lee
Springer, T. J.
Sprunger, Russell
Squires, Stephen
Srini, Vason P.
Sriram, Duvurru
Stallard, James R.
Steinberg, L.
Storey, Thomas F.
Strohl, Mary Jane
Su, Stanley
Sundstrom, Robert J.
Sussenguth, Edward H.
Swartout, William R.
Synwoldt, William J.
Szabados, Michael

Tabak, Mark
Tavs, Jim
Thompson, Bozena H.
Tice, George D., Jr.
Tucker, Mark
Turner, Wendell

Vandendorpe, James
van der Veen, George
Verma, Pramode K.
Vinberg, Anders
Vittal, John
Vohs, Dennis
Voigt, Robert G.
von Mayrhauser, Anneliese
VonSchlegel, Victor, III

Wallace, Bob
Wang, Annie
Ward, Marilyn D.
Watts, H. James
West, Fred
Wheeler, David J.
Whinston, Andrew B.

White, Phil, Sr.
Wholeben, Brent E.
Wohl, Amy D.
Woo, Bob Y.
Woodward, Viola
Worlton, Jack
Wos, Lawrence

Yao, S. Bing
Young, Robert
Young, Wei

Zack, Robert J.
Zloof, Moshe M.
Zvegintzov, Nicholas

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS

President
Stephen S. Yau
Northwestern University
Evanston, IL

Vice President
Per A. Holst
Kulicke & Soffa Industries, Inc.
Willowgrove, PA

Treasurer
Rolland B. Arndt
Sperry Corporation
St. Paul, MN

Secretary
Herbert B. Safford
GTE Data Services, Inc.
Marina Del Rey, CA

BOARD OF DIRECTORS

AFIPS Immediate Past President
Sylvia Charp
Upper Darby, PA

Association for Computational Linguistics (ACL)
Norman K. Sondheimer
USC Information Sciences Institute
Marina Del Ray, CA

Association for Computing Machinery (ACM)
Adele Goldberg
Xerox Corporation
Palo Alto, CA

Michael A. Harrison
University of California
Berkeley, CA

David Kniefel
Deloitte, Haskins & Sells
New York, NY

Association for Educational Data Systems (AEDS)
John Hamblen
University of Missouri—Rolla
Rolla, MO

American Statistical Association (ASA)
Jack Moshman
Moshman Associates, Inc.
Bethesda, MD

American Society for Information Science (ASIS)
James M. Cretsos
Merrell Dow Pharmaceuticals, Inc.
Cincinnati, OH

Data Processing Management Association (DPMA)
J. Ralph Leatherman
Hughes Tool Company
Houston, TX

Carroll L. Lewis
Commercial Data Corporation
Memphis, TN

Donald E. Price
Roseville, CA

IEEE Computer Society
Tse-yun Feng
Pennsylvania State University
University Park, PA

Martha Sloan
Michigan Technological University
Houghton, MI

Stanley Winkler
Bethesda, MD

Instrument Society of America (ISA)
Chun H. Cho
Fisher Controls International, Inc.
Marshalltown, IA

Society for Computer Simulation (SCS)
Walter J. Karplus
University of California
Los Angeles, CA

Society for Industrial and Applied Mathematics (SIAM)
Shmuel Winograd
IBM Research Center
Yorktown Heights, NY

Society for Information Display (SID)
Howard L. Funk
IBM Corporation
White Plains, NY

NATIONAL COMPUTER CONFERENCE BOARD/AFIPS CONFERENCE BOARD

Chairman and AFIPS Representative
Per A. Holst
Kulicke & Soffa Industries, Inc.
Willowgrove, PA

Vice Chairman and SCS Representative
Carl W. Malstrom
North Carolina State University
Raleigh, NC

Secretary and DPMA Representative
George R. Eggert
Institute for Certification of Computer Professionals
Des Plaines, IL

Treasurer and AFIPS Representative
Rolland B. Arndt
Sperry Corporation
St. Paul, MN

AFIPS Representative
Stephen S. Yau
Northwestern University
Evanston, IL

AFIPS Representative
Chun H. Cho
Fisher Controls International, Inc.
Marshalltown, IA

ACM Representative
Seymour Wolfson
Wayne State University
Detroit, MI

IEEE-CS Representative
Russell E. Theisen
Winter Park, FL

ACM President—Ex Officio
Adele Goldberg
Xerox Corporation
Palo Alto, CA

IEEE-CS President—Ex Officio
Martha Sloan
Michigan Technological University
Houghton, MI

SCS President—Ex Officio
Norbert Pobanz
Sunnyvale, CA

DPMA President—Ex Officio
Eddie M. Ashmore
Southern Baptist Theological Seminary
Louisville, KY

NCCC Chairman—Ex Officio
Arnold P. Smith
IBM Corporation
White Plains, NY

Chairman, Industry Advisory Panel
Jim Morris
Publisher, *Datamation*
New York, NY

NATIONAL COMPUTER CONFERENCE COMMITTEE OF THE NCC BOARD

Chairman
Arnold P. Smith
IBM Corporation
White Plains, NY

Secretary
Robert C. Spieker
AT&T Communications
Piscataway, NJ

Members
Morton M. Astrahan
IBM Research Laboratory
San Jose, CA

James F. Foley, Jr.
Life Office Management Association
Atlanta, GA

Floyd O. Harris
Karak Systems, Inc.
Atlanta, GA

Albert K. Hawkes
Sargent & Lundy Engineers
Chicago, IL

Jerry L. Koory
The Rand Corporation
Santa Monica, CA

Donald B. Medley
California State Polytechnical University
Pomona, CA

Hans D. Puehse
Convention Travel Consultants
Novato, CA

William H. Sitter
Tenneco, Inc.
Houston, TX

OAC '86 Chairman—Ex Officio
William J. Carlisle
Bell Communications Research, Inc.
Piscataway, NJ

NCC '85 Chairman—Ex Officio
Karl E. Martersteck
AT&T Bell Laboratories
Naperville, IL

NCC '86 Chairman—Ex Officio
Mary Rich
PFS, Inc.
El Segundo, CA

AFIPS Liaison—Ex Officio
John Gilbert
AFIPS
Reston, VA

AFIPS HEADQUARTERS STAFF

OFFICE OF EXECUTIVE DIRECTOR

Executive Director
Vacant

Executive Assistant (Acting)
Mark Jaede

Executive Secretary
Vacant

Governmental Activities Director
Vacant

Secretary
Vacant

CONFERENCE DEPARTMENT

Director of Conferences
John Gilbert

Administrative Assistant
Cathy Isaacs

Project Manager
Ann-Marie Bartels

Project Coordinator
John Balderson

Operations Manager
Rick Dobson

Senior Exhibit Operations Coordinator
Jill Newman

Conference Operations Coordinator
Sally Gorham

Operations Coordinator
Barbara DeHart

Marketing/Sales Manager
Marty Byrne

Marketing Coordinator
Helen Mugnier

Marketing Coordinator
Michael Sherman

Marketing/Sales Support
Cathy Shippert

Senior Exhibit Sales Coordinator
Katherine Stormont

Exhibit Sales Coordinator
Molly Finney

Exhibit Sales Coordinator
Stephen Gregory

Exhibit Sales Support
Diana Lefcowitz

Registration Manager
Dennis Smoot

Registration Coordinator
Susan Sullivan

Registration Support
Terry DiMurro

COMMUNICATIONS DEPARTMENT

Director
Dianne Edgar

Administrative Assistant
Mary Ford

Coordinator/Editor
Kaylee Jennings

Initiatives Coordinator
Kathryn Potts

Communications Coordinator
Vacant

Communications Support
Vacant

FINANCE/ADMINISTRATION AND DATA PROCESSING DEPARTMENT

Director
William Grubb

Administrative Assistant/Secretary
Gerri Tesh

Accounting Manager
Hope Reynolds

Bookkeeper/Accountant
Pat Whiteaker

Payroll Clerk
Renee Babarsky

Accounting Clerk
Ethel Baltimore

Accounting Clerk
Reem Qubain

Accounting Clerk
Vacant

ADMINISTRATION

Manager
Debra Guazzo

Administration Support
Christopher Powers

Receptionist
Vacant

Maintenance Engineer
Gerald Corbett

DATA PROCESSING

Manager
Richard Ide

Operator/Analyst
Vacant

AFIPS PRESS

Director
Christopher N. Hoelzel

Administrative Assistant
Sharon Osborne

Fulfillment Administrator
Olive Shilland

Editor
Kathleen M. Gagne

NCC Proceedings Production Editor
Elizabeth G. Emanuel

NCC Proceedings Editing/Graphics
Cam Leger
Jack Zibulsky

AUTHOR INDEX