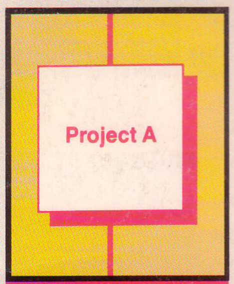
BYTE

NOVEMBER 1988 A McGRAW-HILL PUBLICATION

REVIEWS
Compaq 386s
FlexCache 25386
Mac and PC Transputers
SpinRite
FullWrite
Zortech C++
PC Lint

PRODUCT FOCUS



10 Project Managers

Steve Jobs' new *"machine for the '90s"*

The NeXT Computer

- 25-MHz 68030 • Optical Drive
- Math and Digital Signal Processors
- 8 Megabytes of RAM
- Windowing Unix

IN DEPTH
Parallel Processing

PLUS
Scotland's Innovative Rekursiv Chip
PC Backup Power Supplies
Parallelizing Prolog
5 Short Takes

$3.50 U.S.A./$4.50 IN CANADA
0360-5280

Parallel Processing

Parallel processing could be described as the ultimate in teamwork. In fact, the kind of teamwork involved is not unlike that found in the football stadium on an autumn Sunday afternoon. The quarterback has his job to do, the center has his, the ends and backs have theirs, and the guards and tackles have theirs. All these jobs are under way at the same time, but they're all different and being done by a different player—parallel processing.

Similarly, when a group of people are raking leaves, different people are doing the same job, at the same time, with the result of significantly cutting down on the time required—also parallel processing. Not all jobs, however, can be done in parallel. That Thanksgiving turkey we look forward to at the end of the month can't be rushed—microwaves aside.

The same basic concepts apply in computing. Multiple processors operating in parallel can perform many, but not all, jobs faster than uniprocessors. A logically sequential program must still run sequentially. However, a modular program, or one that can be made modular, can run different sections on different processors and improve its speed.

Last summer, NASA's Jet Propulsion Laboratory introduced the Mark 3 Hypercube parallel supercomputer. Parallel processing has long been the exclusive realm of very large systems; however, it is now becoming available at the microcomputer level. For example, Zenith has announced the Z-1000 with its parallel 80386s (see Microbytes on page 11), and Cogent has come out with the XTM (see the text box "The Crossbar Connection" on page 278).

This month, we look at the world of parallel processing from the microcomputer view. In "Side by Side," Klaus K. Obermeier looks at the field as a whole: the appropriate algorithms and applications; the programming languages, including old favorites and new ones with special parallel-processing functionality; and the hardware and operating-system architectures involved.

One particularly applicable piece of hardware is the transputer board. In "T800 and Counting," Richard M. Stein looks at the T800 transputer board from INMOS, discussing both the hardware aspects of the transputer and the related software aspects of the occam language—the two were designed to work together.

Another language designed for parallel processing on the transputer is Yale University's Linda. In "Getting the Job Done," David Gelernter, one of the language's designers, gives us the inside scoop on the current state of Linda, what it does, how it does it, and its specifically parallel features.

Finally, we have an article on a different way of making computers. In "The Third Dimension," Michael J. Little and Jan Grinberg describe the inner workings of Hughes Research Lab's 3-D Computer. It's an innately parallel computer built not of chips but of *wafers*—stacked wafers. It's a fascinating technology.

While the concept and practice of parallel processing have a history in the large-computer arena, the idea of having parallel-processing power on a desktop is still very new. The Mark 3 Hypercube is intended for simulations for the Strategic Defense Initiative. Can that kind of power really exist on a desktop?

—Jane Morrill Tazelaar
Senior Technical Editor, In Depth

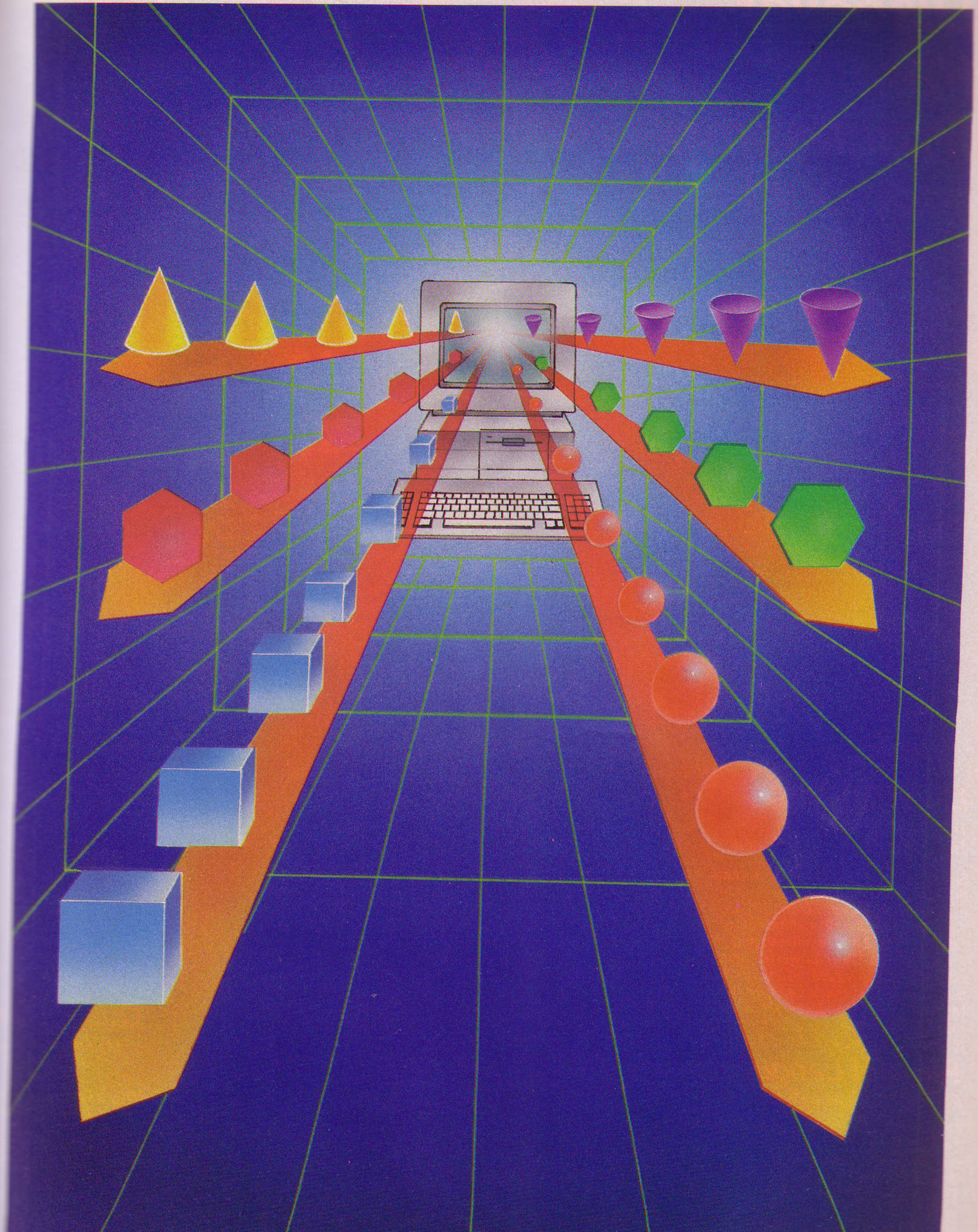


ILLUSTRATION: ROBERT PASTERNAK © 1988

Discover Parallel Processing!

Monoputer/2™

The World's Most Popular Transputer Development System

Since 1986, the MicroWay **Monoputer** has become the favorite transputer development system, with thousands in use worldwide. Monoputer/2 extends the original design from 2 to 16 megabytes and adds an enhanced DMA powered interface. The board can be used to develop code for transputer networks or can be linked with other Monoputers or Quadputers to build a transputer network. It can be powered by the 20 MHz T414 or T800 or the new 25 MHz T425 or T800.

Parallel Languages

Fortran and C Make Porting a Snap!

Microway stocks parallel languages from 3L, Logical Systems and Inmos. These include one Fortran, two Cs, Occam, Pascal, and our own Prolog. We also stock the NAG libraries for the T800 and Rockfield's structural and thermal finite element package. A single T800 node costs $2,000, yet has the power of a $10,000 386/1167 system. Isn't it time you considered porting your Fortran or C application to the transputer?

Quadputer

Mainframe Pow
For Your PC

MicroWay's **Quadputer**
most versatile multiple transputer
on the market today. Each processor can
1, 4 or 8 megabytes of local memory. In addition,
or more Quadputers can be linked together with
cables to build large systems. One MicroWay customer
an 8 hour mainframe analysis to 15 minutes with five Quadput
giving him realtime control of his business.

For further information, please call MicroWay's Technical Support staff at (508) 746-7341.

MicroWay

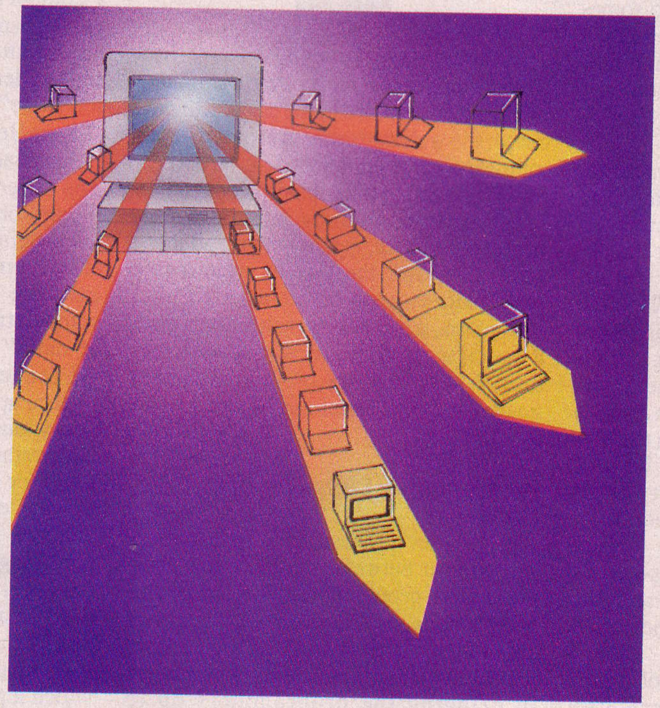
World Leader in PC Numeric

P.O. Box 79, Kingston, MA 02364 USA (508)
32 High St., Kingston-Upon-Thames, U.K., 01-541-
USA FAX 617-934-2414 Australia 02-439-8400 Germany 069-75-

Side by Side

You can only simulate true parallelism on your personal computer today, but tomorrow will be another story

Klaus K. Obermeier



A parallel-processing computer, simply defined, is one that can perform operations using more than one processor simultaneously. You can generally divide parallel processing into three major areas of research: algorithms and applications; programming languages; and architecture, including hardware and operating systems.

Where to Start

The conventional serial computer suffers from one serious drawback: the way the CPU accesses memory. While data is being retrieved from memory, it is actually written into a processor register, and after the register is incremented, the new value is put back into memory. During this period, the CPU remains idle. This phenomenon, known as the von Neumann bottleneck, accounts for the sometimes slow and inefficient use of conventional serial-processor resources.

But parallel processing has been around longer than the von Neumann bottleneck. As early as 1840, Charles Babbage conceived of a way to perform multiplication and indexing arithmetic simultaneously. The first operating parallel processor was the ILLIAC IV. This machine, developed by Dan Slotnick in 1966 at the University of Illinois, featured 64 processors.

Although the first commercial parallel-processing system flopped—the $7 million Heterogeneous Element Processor, developed in 1985 by Denelcor—by 1986 more than a dozen companies were either selling or in the process of building parallel processors, including Bolt Beranek and Newman, Cray Research, DEC, IBM, Intel, Alliant, Encore, and Thinking Machines.

Today, parallel-processing systems, such as the Connection Machine from Thinking Machines, can execute a few billion operations per second using up to 65,536 processors simultaneously. Searching a database of over 30,000 documents (18 megabytes) on a 16,384-element Connection Machine takes about 0.004 seconds for a Boolean query with 25 terms. Dow Jones recently purchased two 32,000-processor, 256-megabyte Connection Machines for use with its information-retrieval services.

The Parallel Approach

The central problem parallel-processing systems face is how to effectively and efficiently use more than one processor at the same time. A system's effectiveness depends on whether you can identify a problem that lends itself to parallelism, determine the algorithm, and map it onto a suitable architecture.

As you can imagine, problems arise if more than one processor requires access

continued

ILLUSTRATION: ROBERT PASTERNAK © 1988

to the same memory location or if more than one processor tries to increment data in the same memory location. Therefore, the common argument that more processors are always faster than one holds true for systems that can cope with problems such as contention and have appropriate synchronization mechanisms in place.

Another factor that can prevent successful use of parallelism is the bottom-up approach parallel-processing-system architects often take to hardware design. Simply put, they sometimes don't consider the needs of the application designer when they configure the hardware. People who write parallel applications should always keep in mind the target architecture so they can be sure their application-design algorithm will be suitable (e.g., whether they will use message passing or shared memory).

The use of parallelizing compilers is no answer to this problem. Parallelizing compilers are most suitable when past investment does not warrant rewriting the existing software. The programmer has to consider the problem from two sometimes opposing points of view: top-down for the design of the algorithm and bottom-up for the actual implementation.

Algorithms and Applications

Parallel processing's most common applications are simulation, modeling, and optimization programs for commercial use. Airline scheduling is among the potential applications—calculating seat assignments and about 200,000 to 250,000 necessary changes in routing daily takes United Airlines' current aircraft assignment model 15 hours of CPU time. If you were a programmer faced with such a task, you would first break down the task into sizable chunks that could be processed in parallel and then worry about synchronization between the processors. Unfortunately, your creativity for designing a solution would be hampered by the existing operating system and the idiosyncratic architecture of the target hardware.

What you should first do in such a situation is decide the necessary *granularity* of the application. Granularity refers to the amount of time being spent on communicating versus computing in a parallel program. In a coarse-grained application, the parallel-processing system consists of large independent chunks with little time—on the order of hundreds of communications per second between processors—spent on communicating between the individual processors. In a fine-grained application, more time—millions of communications per second—is spent on communicating and synchronizing between the processors. In any case, you have to leverage your solution with the encountered architecture. In the example of the aircraft assignment task, a processor may be assigned to one flight in a fine-grained system and to an entire aircraft in a coarse-grained system.

Once you determine the application's granularity, consider what form the communication between processors should take, via shared memory or message passing in distributed systems. While processors in a shared-memory system communicate via a common data structure, message passing takes place between two processors. Ultimately, atomic operations (e.g., locks, semaphores, and monitors) take over the task to synchronize processors properly. Once the algorithm is in place, it has to be synchronized with a specific parallel-processing architecture. What role do the programming languages play in matching algorithms to architectures?

People who write parallel applications should always keep in mind the target architecture.

Programming Languages vs. Compilers

For parallel processing, programmers can choose between parallelizing compilers and genuinely parallel programming languages. Parallelizing compilers are often used because of the high investment in existing application software or in bringing the programmers up to speed. As with digitizing old recordings, parallelizing serial algorithms doesn't work as well as algorithms genuinely conceived for parallel implementations. And as for the alleged complexity of writing programs in parallel, according to Chuck Seitz, chief designer of CalTech's Cosmic Cube, the precursor to Hypercube, all other things being equal, "Programming experimental computers like the Cosmic Cube is not much harder than programming sequential computers, if the problem lends itself to a concurrent solution."

According to David Gelernter of Yale University, parallel-programming languages can be classified into three categories: Algol-based languages (e.g., Ada, Linda); parallel Lisps and logic languages (e.g., Multilisp, Concurrent Prolog); and parallel functional languages (e.g., Parafl).

Algol-based languages span the spectrum from Ada, originally designed for systems that execute sequentially, to Linda, especially conceived for parallel processing. Whereas Ada includes tools to support parallel processing, a parallel program written in Linda is, according to an article by S. Ahuja, a "spatially and temporally unordered bag of processes, not a process graph."

The Linda system consists of operators that can turn any host language (e.g., FORTRAN or C) into a parallel-programming language. However, it is an autonomous language consisting of a run-time kernel for synchronization and a compiler. Furthermore, it allows for parallelism both in the form of partitioning simultaneous processes and in replicating identical ones. (See "Getting the Job Done" on page 301.)

Parallel Lisps focus on symbolic rather than numeric parallel processing. The difference between the two, according to Robert Halstead, is that "numerical programs may be described as delivering numbers to an arithmetic unit to calculate a result," whereas "symbolic computation emphasizes rearrangement of data." Consequently, parallel Lisps are prime candidates for artificial-intelligence applications with an emphasis on operations such as recursion on trees and lists, rather than iterations in the form of loops for numerical computations.

Parallel functional programming includes a methodology that allows mapping programs to parallel-processing topologies. According to Paul Hudak, the most important aspect of this methodology is that "it treats the multiprocessor as a single autonomous computer onto which a program is mapped, rather than as a group of independent processors that carry out complex communications and require complex synchronization." Rather than having side effects from assignment statements, functional languages guarantee that a program will have the same result regardless of the order in which it has been executed. Therefore, in functional languages the parallelism is implicit and supported by

continued

The Crossbar Connection

Frank Hayes

Two processors should be twice as fast as one, and a thousand processors should be a thousand times as fast—at least in theory. But that depends on getting those thousand processors talking to each other, and that's not an easy task. In a fully connected network, interconnecting only 30 processors requires 435 separate connections; 1000 processors would require 499,500 connections.

A fixed network structure (such as a Hypercube) avoids that problem by connecting each processor to only a few neighbors, but then data must be passed from processor to processor through the network for distant processors to communicate. Alternatively, all the processors can share a common communications bus, but that risks tying up the bus if two processors have lots of data to exchange, bringing communications for the rest of the system to a screeching halt.

Cogent Research in Beaverton, Oregon, thinks there's a better solution. Cogent's new desktop supercomputer, the XTM (see photo A), can connect any number of parallel processors, without passing data hand-to-hand through a network or tying up a common bus when there's lots of data to exchange. Instead, the Cogent machine has a hybrid communications architecture that has both a common bus and a unique network system.

The Cogent XTM

The XTM's processors are INMOS T800 transputers. Each transputer has 4 megabytes of RAM, as well as four high-speed serial-communications channels specifically designed for exchanging data with other transputers.

Photo A: *Cogent Research's XTM parallel desktop supercomputer, based on the INMOS T800 transputer and Yale University's Linda programming language. (Photo courtesy of Cogent Research, Inc.)*

In the XTM, the transputers all share an ordinary parallel-communications bus, through which messages can be sent. Separately, the four serial-communications channels from each transputer are connected to an intelligent switching system. Inside the intelligent switch, the serial-communications channels from all the transputers in the system are arranged in a network—but with no permanent connections. Upon request, the intelligent crossbar switch can directly connect any two transputers in the network. Consequently, any two transputers can talk either through the shared bus or through a temporary "private" direct connection (see figure A).

For example, suppose processor A wants to send a large collection of data to processor B. If A sent the data through the common bus, it would tie up the bus—a classic communications bottleneck. Instead, A sends a message through the bus to the crossbar-switch controller, asking for a direct connection to B.

Once the connection is made, A can send data to B at high speed without interfering with any other processor's communications. Once the data transfer is complete, A sends another message to the switch controller, asking it to disconnect A from B, and the two processors are free to make new connections.

Meanwhile, every other pair of processors in the system can be connected in the same way. While A and B are exchanging data, C and D can make their own connection. At least in theory, in a 1000-processor system, 500 serial connections could be transferring messages at high speed.

It takes the XTM's intelligent crossbar switch less than 40 microseconds to link any two processors, and only 200 to 400 μs to completely reconfigure the entire computer. (The XTM can even be reconfigured to mimic a Hypercube or another fixed network.)

Because the communications network is dynamically reconfigurable, all the processors can communicate directly through a relatively small number of communications channels. One thousand processors can communicate using only 4000 serial lines—less than 1 percent of the number required for a fully connected network. As a result, the number of processors in the system is almost unlimited. Cogent has designed a system for Sandia National Laboratory that contains 1900 processors and has roughly the same computing power as a Cray X-MP.

A more typical Cogent XTM system sits on a desktop and has two processors in a workstation cabinet that's slightly smaller than an IBM PC (14 by 14 by 6 inches). Along with the processors, the workstation contains a 90- or 190-megabyte hard disk drive, an 800K-byte 3½-inch floppy disk drive, and three NuBus slots.

There are also an external 1024- by 808-pixel display, a keyboard, and a mouse. The least expensive XTM system (with a 90-megabyte hard disk drive and a monochrome display) costs $19,800.

To add processors to this basic system, you first need to add a resource server (a 14 by 18 by 6 cabinet with 16 slots and the intelligent crossbar switch) and a communications card to connect it to the XTM.

As a result, going from two processors to four adds another $35,000 to the price. After that, you can add computation cards (each one contains two transputers) for $12,000 each—until you run out of slots in the resource server, at which time you can add another resource server. Additional disk storage comes in the form of a disk server (1.9 gigabytes, plus an 810-megabyte optical drive for backup, for $60,000). The workstation, resource servers, and disk servers all communicate through fiber-optic cable at 100 megabits per second.

An Easy Growth Path

The price on a desktop supercomputer can rise quickly. A workstation with a single resource server packed full of processors fits easily on a desktop—and costs over $200,000. The Cray-class system Cogent designed for Sandia will cost $15 million to build.

But the XTM is unique among supercomputers in that both the two-processor minimal system and the 1900-processor Sandia machine use exactly the same hardware. And with enough time and money, you can build any system into a colossus—without changing the software. The XTM's operating system is based on the Linda parallel-programming concept, which is effectively blind to the number of processors in the computer. A program written in FORTRAN or C using the Linda extensions will run on a minimal XTM system. Add two (or a dozen) more processors, and the program will run in exactly the same way—but nearly twice (or a dozen times) as fast.

And how fast is fast? Each of the XTM's transputers adds 3 million floating-point operations per second of processing power. Cogent's designers believe that because the XTM can be so easily tailored to match computational problems—adding more number-crunching capability as it's needed—the new machine will open up a completely new range of problems that were previously inaccessible from desktop workstations.

Frank Hayes is an associate news editor for BYTE in San Francisco. He can be reached on BIX as "frankhayes."

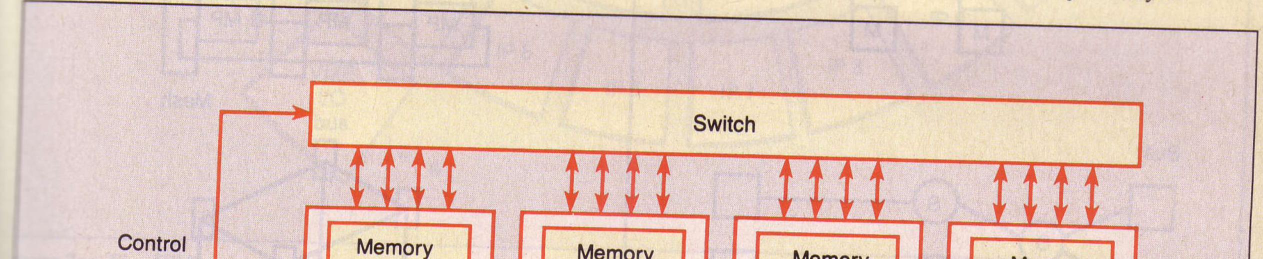


Figure A: *The Cogent XTM's communications system. For A to exchange data with B, A notifies the switch controller via the communications bus. The controller then orders the intelligent switch to make a direct connection between A and B.*

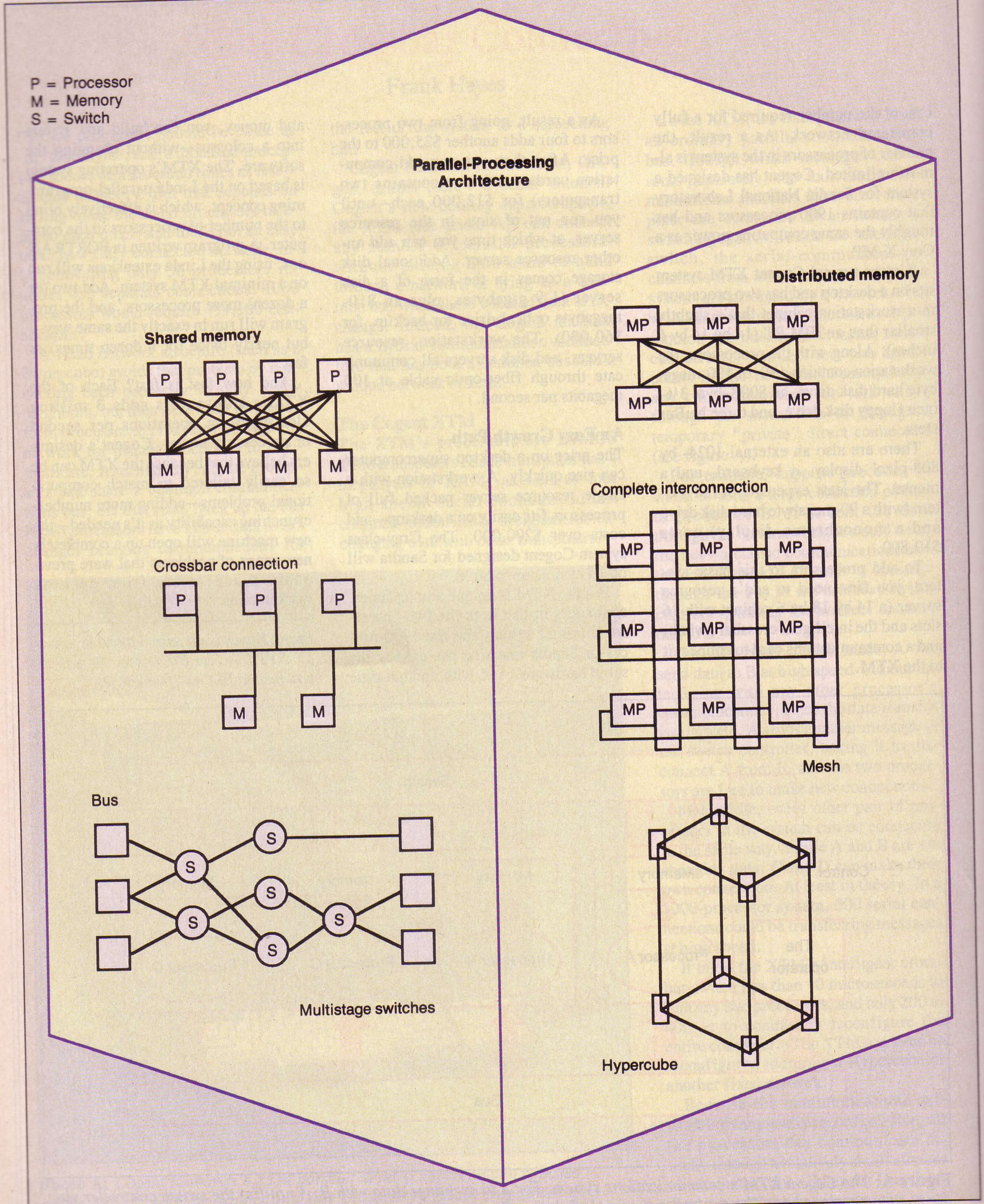


Figure 1: *Shared memory architectures allow parallel systems to access a common shared memory, as opposed to distributed memory systems, which provide memory to each processor.*

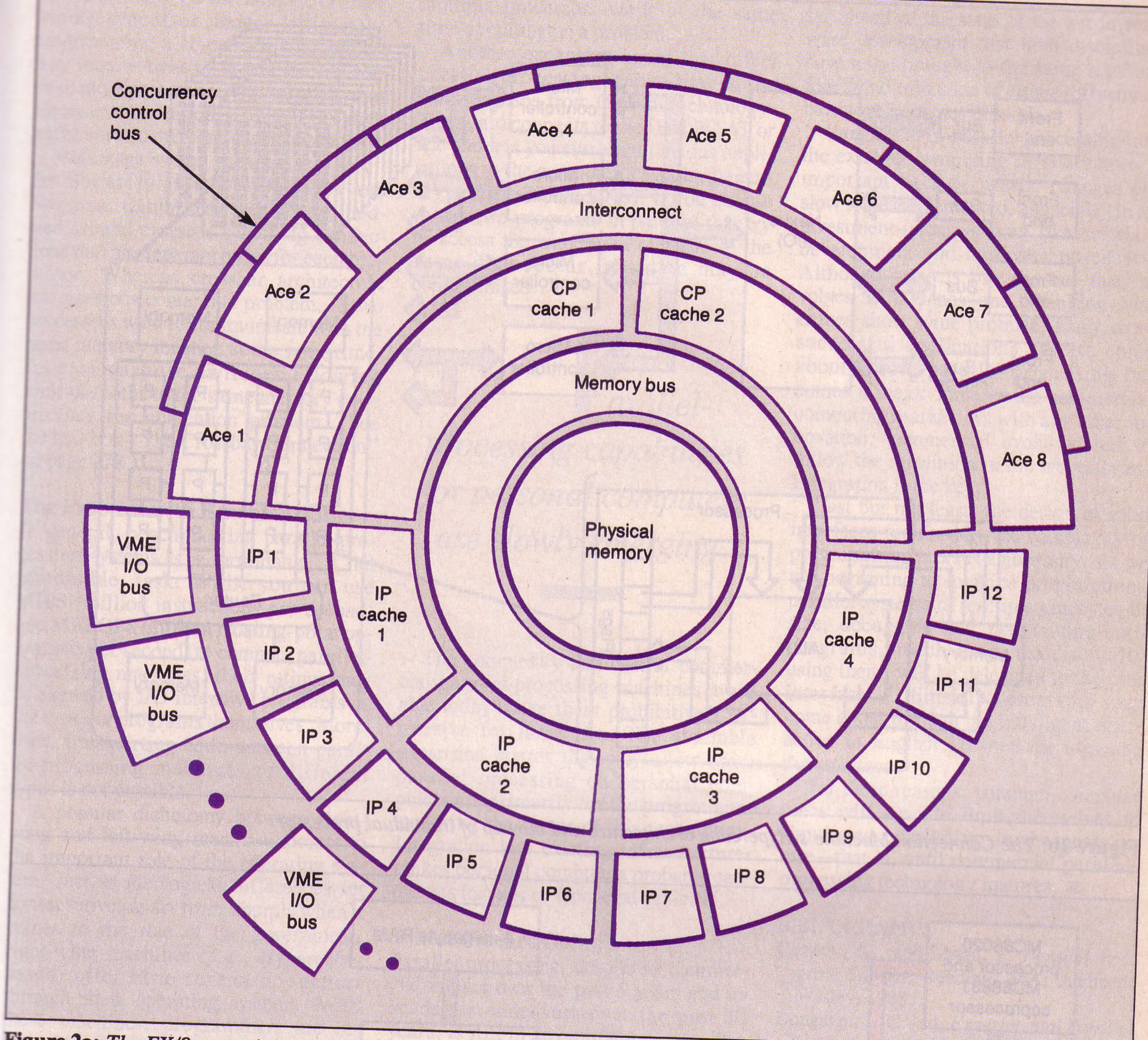


Figure 2a: *The FX/8 system's common-bus architecture. The ACE processors (Advanced Computational Elements) function as application processors. I/O and operating-system jobs are performed by the interactive processors, which are 68020-based.*

underlying semantics. In brief, functional languages are prime candidates for programming parallel machines.

All about Architecture

A dichotomy exists between shared-memory architectures and distributed-memory architectures. The former allow parallel systems to have access to a common, shared memory, while the latter give each processor its own memory. As shown in figure 1, a multitude of configurations is possible and has been used for building parallel-processing systems.

The most widely used architectures are bus-based systems, the Hypercube, and a design using special-purpose switches, as shown in figures 2a, 2b, and 2c. A less widely known architecture is found in wafer technology. (See "The Third Dimension" on page 311.)

Bus-based systems (e.g., the FX/8, from Alliant Computer Systems, Littleton, Massachusetts) provide the simplest form of parallelism, having a set of processors connected to a set of memory boards via a common bus. Although these systems are attractive for their simplicity, problems arise in the form of limited scalability, contention for accessing the same memory location, and rising costs for overall speed gain. In applications dominated by scalar code, like older mechanical CAD applications, you'll find good speed increases with from one to four processors, but after that, adding processors won't increase speed.

Hypercube (e.g., the Connection Machine from Thinking Machines, Cambridge, Massachusetts), based on CalTech's Cosmic Cube, allows multidimensional connections between processors, thus connecting every processor at least indirectly. Although it's attractive for its capability to interconnect thousands of individual processors, communication speed between processors may

continued

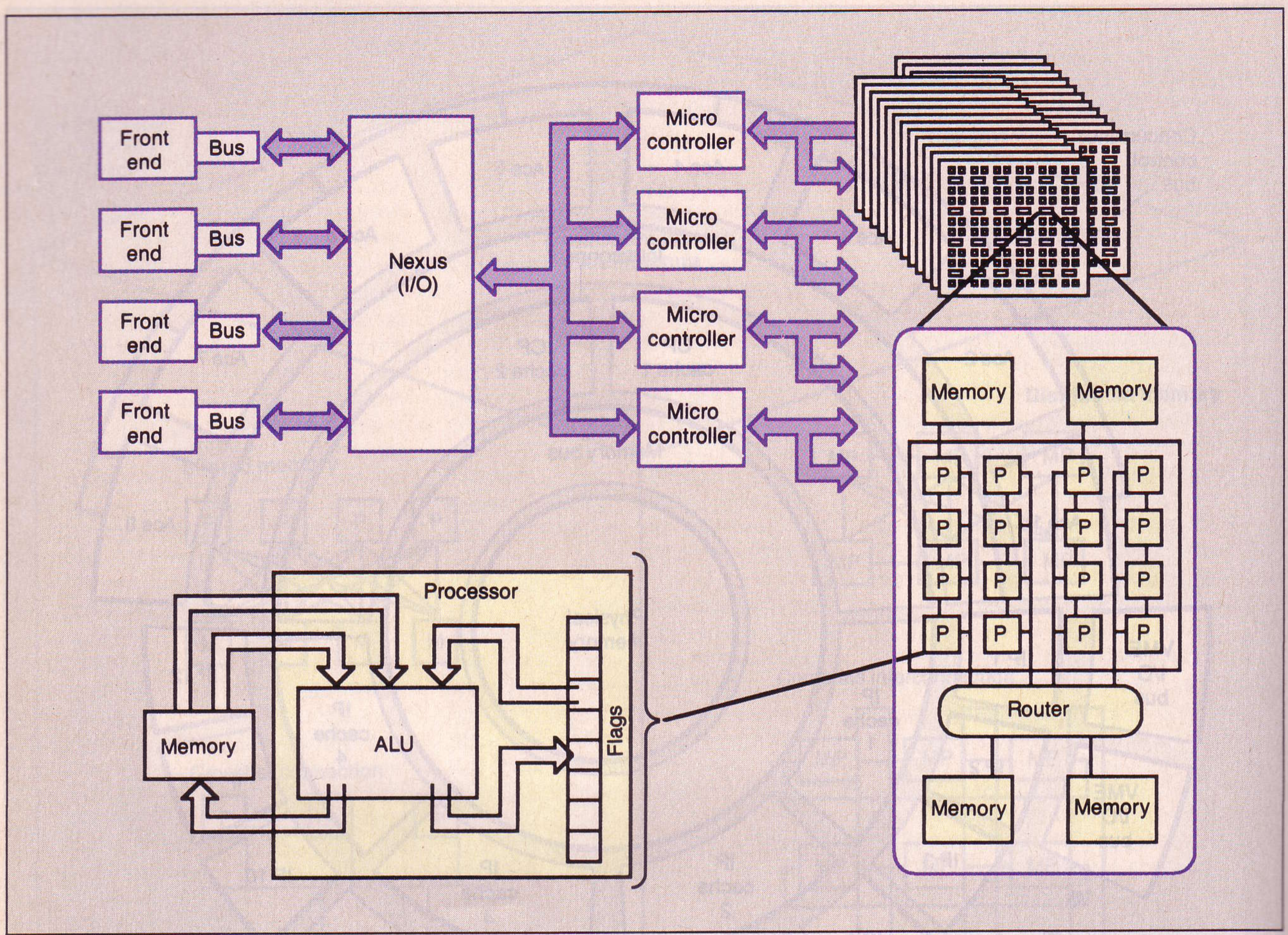


Figure 2b: *The Connection Machine's Hypercube architecture and blowup of individual processor.*

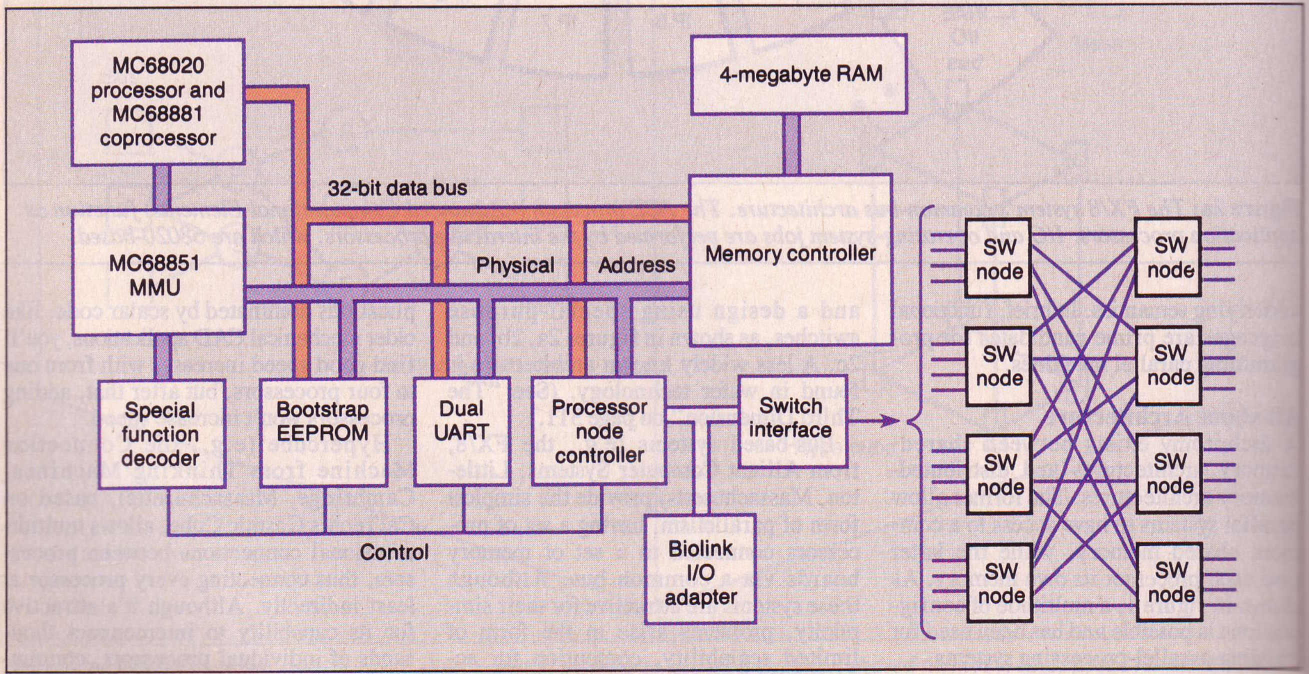


Figure 2c: *A Butterfly processor node block diagram and blowup of the Butterfly switch design.*

vary depending on the number of intervening processor nodes. Ultimately, programming a Hypercube architecture may require treating it as a loosely coupled multiprocessor with explicit data placement and task assignment per node under certain memory size limitations.

Multistage switch architectures (e.g., the Butterfly from Bolt Beranek and Newman, Cambridge) are closely modeled around crossbar switching connections that use separate buses for each processor. Whereas crossbar architecture has a serious contention problem, if two processors want to communicate with the same memory location at the same time, special-purpose switches allowing multiple paths to the same memory node alleviate the contention problem. (See the text box "The Crossbar Connection" on page 278.)

The Problem with Benchmarking

In general, benchmarking parallel-processing systems is a formidable, if not impossible, task. While you can use MIPS (million instructions per second) and MFLOPS (million floating-point operations per second) to compare parallel-processing machines, their ratings may be skewed by I/O-intensive programs or the types of programs themselves. Moreover, transporting code between parallel-processing machines of different types is not possible.

A popular dichotomy between right-wing and left-wing machines points to the important role of the operating system. Just as ideological differences set conservatives apart from liberals when it comes to the role of the government, right-wing machines (e.g., Hypercube-based) offer little control or support through their operating systems. With these machines, programmers are required to code more low-level operations themselves. Left-wing machines (e.g., Butterfly) provide more generous support by the operating system.

The Personal Computer Connection

While parallel processing has reached the minicomputer market, parallel-processing capabilities for personal computers are slowly emerging and usually come in two forms: expansion boards and software simulation. (See "T800 and Counting" on page 287.) The INMOS D700 Card and T414 CPU offer IBM PC users parallel-processing capabilities as part of the D701 Transputer Development System. The package comes with the Algol-based programming language occam. The T414 CPU, a 32-bit microprocessor, is able to pass information to multiple processors while at the same time operating on a problem.

Another expansion board that offers parallel-processing capability is the PCturbo 286e from Orchid Technology. The board connects to the IBM PC AT or XT and lets you run simultaneous applications in the computer's standard memory and the 286e's RAM. If you choose to run two programs in parallel that try to access the same data or write to the same disk sector, the data may be compromised.

Parallel-processing capabilities for personal computers are slowly emerging.

The complexity and cost of commercial parallel-processing machines available today make them prohibitively expensive for mass use. The available expansion boards that may allow some parallel processing on personal computers are primarily for the programmer exploring the flavor of this technology. Integration into existing infrastructures makes personal computers probable candidates for host or front-end vehicles.

Where We Are Going?

Parallel processing, despite its commercial impact over the past 5 years and its academic endeavors over the past 20 years, is still in an embryonic state. The three most significant issues standing in the way of necessary commercial breakthroughs are standardization of parallel-processing languages and architectures, integration of parallel processing into the existing computing infrastructure, and design of solid interfaces required by the complexity of the programming tasks.

Standardization of parallel-processing languages and architectures is important because there is currently no standardization of parallel-processing techniques in sight. One reason for this is that parallel-processing technologies are still mostly in R&D laboratories, an environment that promotes individualism. Another reason is the potential for diverse applications that makes it impossible to predict what such standards should look like. Finally, the design of hardware is so far ahead of the state of the art in software development that both disciplines have to be brought to the same level before standardization of either software or hardware becomes possible.

Integration of parallel processing into the existing computing infrastructure is important because drastic changes are slow to be implemented, especially in an investment-intensive area like software development and hardware purchases. Although projects like the one that involves Dow Jones and Thinking Machines show some promise, many more successful applications have to come about before parallel processing becomes a major force in the commercial computing market. As with any other innovation, commercial evolution has to follow the revolution in the laboratories. Integration is the key.

Last but not least, the design of solid interfaces required by the complexity of programming tasks is a necessity. As we are beginning to imagine programming parallel systems of multiple gigabytes in size, debugging and maintaining such programs, much more than actually using them, will be delegated to the user interface. Ultimately, programs may come about through the intelligent interaction in English between the user and the interface.

For these reasons, personal computer users will have to limit themselves to simulating parallelism in the near future—that is, until commercial parallel-processing technology matures. ■

BIBLIOGRAPHY

Chandy, K., and Misra, F. *Parallel Program Design*. Reading, MA: Addison-Wesley, 1988.

Dongarra, J. C., ed. *Experimental Parallel Computing Architecture*. Amsterdam: Elsevier Science Publishing Co., 1987.

Fox, G. C., and Messina, P. C. "Advanced Computer Architectures." *Scientific American*, October 1987, pp. 66–77.

Frenkel, K. A., ed. "Parallel Processing." *Communications of the ACM*, vol. 29, no. 12, December 1986, pp. 1168–1239.

Gelernter, D., ed. "Domesticating Parallelism." *Computer*, vol. 19, no. 8, August 1986, pp. 12–72.

Gullo, K., and Schatz, W. "The Supercomputer Breaks Through." *Datamation*, vol. 34, no. 9, 1988, pp. 50–63.

Hillis, W. D. *The Connection Machine*. Cambridge, MA: The MIT Press, 1986.

Klaus K. Obermeier is a projects manager of the AI Group at Battelle Laboratories in Columbus, Ohio. He can be reached on BIX c/o "editors."