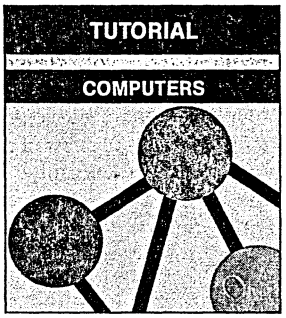
A parallel architecture comes of age at last

Hypercube machines reach the market on the promise of executing billions of operations a second to solve difficult simulation problems



Scientists and engineers have grown to expect the performance of their computers to increase by an order of magnitude about every five years. But that dizzy pace has slowed recently, and supercomputers built around a single processing unit—the Cray-1, the NEC SX-2, or the Fujitsu VP200—may already be within an order of magnitude of their technological limit. This theoretical upper boundary, some 3 gigaflops (billions of floating-point operations per second), is established by the length of time it takes electrical signals to propagate, traveling through the wires at about half the speed of light.

Future scientific and engineering problems, however, in such fields as fluid dynamics, computational chemistry, geophysical modeling, and aerodynamics, are expected to require processing rates far in excess of that 3-gigaflops limit. But by dividing applications among many processors working in parallel, rates in the teraflops range—trillions of floating-point operations per second—are in theory possible. Computers based on tens and even hundreds of processors have already been built, and intensive work is now under way to build practical machines that can be expanded economically to the point where thousands of processors are all working on a problem simultaneously.

While such machines will deliver the computing power needed for problems well beyond the reach of today's machines, they will present new programming tasks. In particular, the break with the shared-memory model is a key consideration for some architectures, because the physical locations of elements of data among the processors must be managed by the programmer. In a shared-memory computer, on the other hand, all processors access a common memory and communicate with each other through messages left in that memory. Furthermore, all the processors have access to all information in the memory.

The most popular architecture for large-scale parallel computers—intended for the kinds of applications normally handled by supercomputers—is the hypercube topology. This architecture has been the subject of at least two research and five commercial ventures since it was first demonstrated at the California Institute of Technology just four years ago [table, p. 49].

More than 100 hypercubes are already in use, most of them in academic institutions and government laboratories. Researchers are learning how to use the technology, determine where it is applicable, and develop software tools to support programming. They have stimulated interest in hypercubes in the industrial, research, and defense sectors, where pressure to keep abreast of new technology is strong.

Hypercubes run multiple programs that operate on multiple sets of data. Within the machine, the individual processing units, called nodes, are independent and communicate with each other while executing programs. Each node has its own memory, floating-point hardware, communications processor, and copy of the operating system and applications program.

The computers are called hypercubes because their architecture can be thought of as a cube of any dimension, with a node at each "corner." The higher the dimension, the more nodes there are. For example, a two-dimensional hypercube takes the form of four nodes connected by communications lines to form a square. In a three-dimensional architecture, eight nodes are connected into a cube. The number of processors is always a power of 2, the exponent representing the hypercube's dimension.

That dimension also denotes the number of other nodes to which each node is directly connected. For example, a six-dimensional hypercube has 64 nodes, each connected by dedicated communications channels to the six closest nodes, which are called its nearest neighbors. A node can communicate with other nodes that are not nearest neighbors only by passing messages through intermediary nodes.

The higher a hypercube's dimension, the higher its communications capacity relative to its computational capacity. For example, a seven-dimensional hypercube has 128 nodes, twice the 64 of a six-dimensional machine. But the seven dimensions provide 896 communications channels (seven for each of the 128 nodes), two and one-third times the 384 channels available in six dimensions.

The hypercube's communications system and each node's individual memory are key characteristics that allow engineers to expand these computers far beyond most parallel architectures. In many parallel computers, processing units share buses and memory, which generally accommodate no more than about 20 processors. Hypercubes, on the other hand, have already been built with 1024 32-bit processors, and machines with 16 000 nodes and more are planned for within five years.

Although one node communicates directly only with its nearest neighbors, it can send a message to a more remote node in hops from neighbor node to neighbor node, on to the destination. The

Defining terms

Explicitly parallel: a programming application that can be executed on multiple processors only with communication between neighboring processors.
Host: a separate processor in a parallel computer used for administrative functions such as input and output.
Node: a processing unit in a concurrent computer, especially one whose communications network is arranged in a hypercube topology; nodes typically include memory, a communications processor, and sometimes a mathematical coprocessor.
Perfectly parallel: a programming application that can be executed on multiple processors without communication between those processors.

Paul Wiley Intel Scientific Computers

0018-9235/87/0600-0046$1.00©1987 IEEE

relaying is supported by each node's operating system. The most hops that a message needs to get from one node to any other is again equal to the hypercube's dimension.

In a six-dimensional system, for example, the longest possible communications path would comprise six hops, while the average length of random communications would be only three hops, assuming that there was no contention between different processors for the same communications channel. Correspondingly, in an eight-dimensional hypercube the average communication would be four hops, even though there would be four times as many nodes altogether as in the six-dimensional system. In general, the average number of nodes in a hypercube's communications paths increases slowly, relative to the increase in its computational capacity, a feature especially critical for large networks of, say, hundreds or thousands of nodes.

With the first, experimental hypercubes, programmers had to manage all communications paths between nodes. Today most hypercubes have operating systems that allow programmers to specify which node is to be a message's destination, without having to specify the message's path, again critical in large, complex applications. In applications designed to run on hundreds of nodes, programming all communications by hand could easily overwhelm all other aspects of the project.

Because they support efficient communications between nearest neighbors, hypercubes work well in the kinds of scientific and engineering applications that involve simulations of inherently concurrent phenomena. Such applications typically call for many independent and simultaneous processes. Adjacent processes affect one another most strongly, so interaction is largely limited to nearest neighbors. That model of loosely connected localized phenomena is closely matched by a hypercube's capabilities.

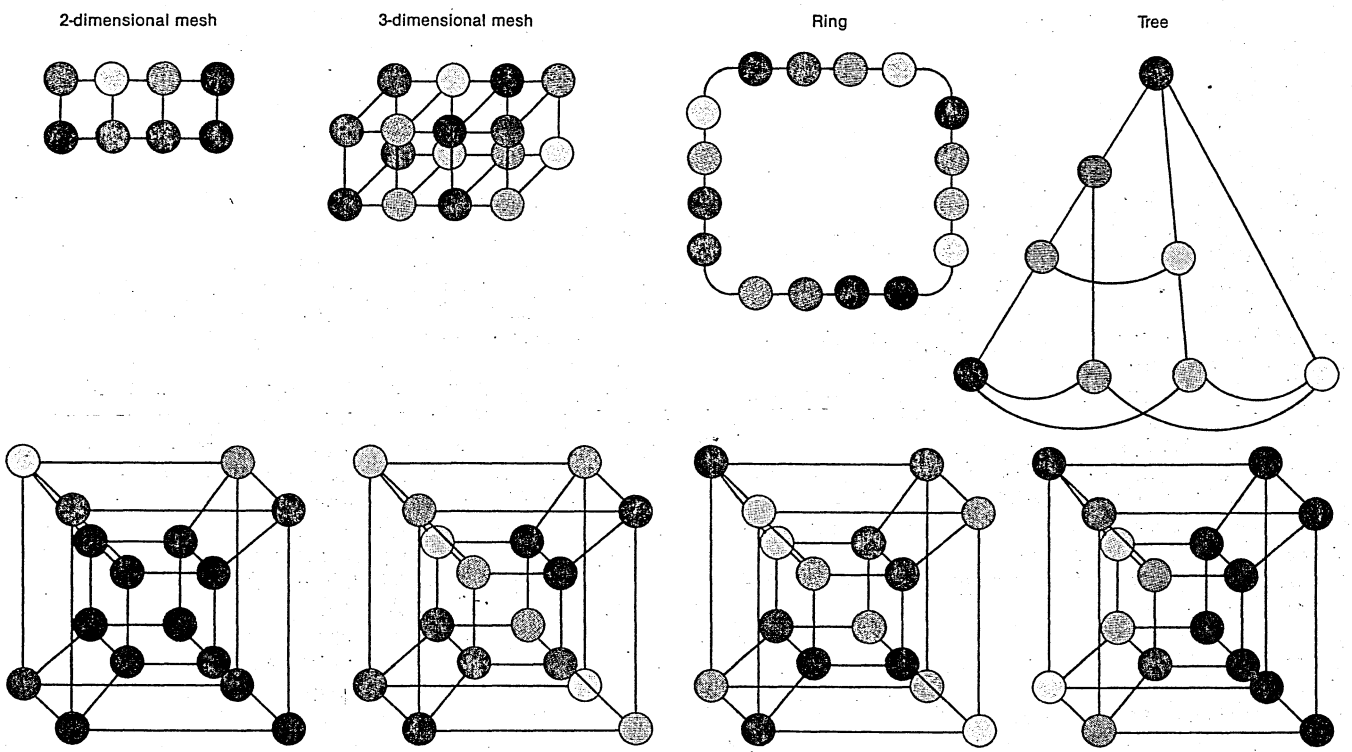
For the sake of versatility, however, future hypercubes will need to support more efficient communication between all nodes in the system. In particular, the software used to route messages between nodes is slow and needs to be replaced with hardware. Several hypercube makers are working on such systems.

Computing in parallel

There are a number of ways to classify parallel applications; the common ones use such programming features as communication characteristics, data-distribution methods, and mathematical techniques. Among the problems classified by communication characteristics are those requiring no communication between nodes, and those requiring communication only between nearest neighbors; the former are sometimes called perfectly-parallel, the latter, explicitly-parallel problems.

Explicitly-parallel applications include simulations of physical phenomena, such as the heat flow on a two-dimensional metal plate. A hypercube represents the plate as a collection of temperatures in individual regions. With a 1024-processor computer a programmer divides the plate into a grid 32 segments by 32. Each processor simulates the temperature distribution within a segment and, as the system converges on a solution, periodically communicates its results to processors simulating adjacent segments. This ensures consistent values along the boundaries. Other applications lending themselves to this type of solution include calculating diffusion of dopant molecules in a semiconductor, and modeling stresses within a solid.

Many such explicitly-parallel applications can be divided among processors in a physical representation of the original data. In a hypercube simulating the meteorology and atmospheric chemistry behind the formation of acid rain in the atmosphere, each node simulates conditions within a subsection of the region of the atmosphere under scrutiny. Neighboring nodes represent neighboring subsections of the atmosphere. Such an application



One attraction of hypercube computers is that their flexible communications network lets programmers choose different topologies for different applications. For example, a four-dimensional system, with 16 nodes, can be treated as a two-dimensional mesh, a three-dimensional mesh, a ring, or a tree (top, left to right) merely by directing communications appropriately between nodes. Color coding of nodes relates each topology—meshes, ring, and tree—to the diagram below it, to show how each would be mapped into a four-dimensional hypercube. Different topologies work best for different kinds of application. Meshes do well for simulations of phenomena in two or three dimensions, such as heat flow or gaseous diffusions; the ring is ideal for determining the effects of each element in a group on the others—for example, the gravitational interrelation of a number of celestial bodies; the tree topology is useful where a value in one node needs to be "broadcast" to all the others.

is explicitly-parallel because simulation of events in one region depends on the status of nearby regions—for example, whether acid-forming pollutants are being emitted, which way the wind is blowing, and so on. Such information is exchanged between nodes for each time interval in the simulation.

In a perfectly-parallel application, each node is regarded as an independent computational unit. On a conventional computer with a single processing unit, such applications run sequentially, one segment after another. On a parallel machine, each processor solves its part of the problem independently. Overall execution time falls by a factor proportional to the number of processors.

Virtually any computation involving a mathematical series is perfectly parallel, because each processor handles a different interval within the series, independently of the other processors. But even these applications are not completely perfectly parallel, of course, because after the computations are performed the processors do need to communicate with each other briefly, to combine the results into the final answer.

As a simple illustration, a method of computing π is integrating, from 0 to 1, the function 4 divided by the square of the sum of 1 and x. Like any integral, that equals the area under the plot of the function, which can be approximated as a series of rectangles. The value of π is estimated by adding up the areas of all the rectangles. Computing π to 12 decimal places requires a series of about 1 million elements.

A conventional, single-processor machine begins by reading the input data, in this case a single integer designating the number of series elements to be computed. Then the processor finds the midpoint of each rectangle used to compute π, evaluating the function at those points and adding all the function values. Next the sum is multiplied by the step size (the width of the rectangles) to approximate π. This simple program actually takes several minutes of superminicomputer time when the number of series elements is 1 million.

The same work can be done on a hypercube. In that case, two programs are written: one for the nodes, the other for the host—the processing unit in the system that handles input and output for the nodes. Programmers interact with the hypercube by way of the host, which reads the input and passes it on to the nodes.

The host loads the program for the hypercube into each node, since this is a perfectly-parallel application with each node performing the same computational task. Each node begins the execution of its program by determining the dimension of the hypercube in use, since the hypercube may be a subset of nodes of a hypercube of higher dimension. Each node determines its own identification number within the hypercube and waits for the host to transmit the number of series elements the nodes will compute to estimate π.

Then the main, computationally intensive portion of the program begins. Each processor executes a loop, computing a subset of the 1 million series elements. If there are 128 nodes, for example, half compute 7812 elements, the other half, 7813. Each node begins by computing an element indexed to its own identification number, and some number of additional elements. For instance, node number one might compute the first, second, and third series elements; node number two the fourth, fifth and sixth, and so on.

To go one step farther, some mathematical series have elements whose computational requirements vary with the index number. Computing an element, for example, might require taking the factorial of the index number, in which case computing the first element would require fewer operations than computing the second, which would require fewer than computing the third, and so on.

In such a case, balancing the computational load becomes an important consideration for hypercube programmers. If 100 elements in a series of this type were to be computed by a four-node hypercube, with the first node taking the first 25 elements, the second node the next 25, and so on, the load would be poorly balanced. The first node would do much less work than the fourth, which would take more time than the other three. Redistributing the work would solve the problem, with the first node taking the first element, and every fourth element after that; the second node would take the second element, and every fourth after that, and so on for the other two nodes—rather like the way cards are dealt in a game of bridge.

Once each node has completed its share of the processing and summed its elements, all the partial results are added. The final answer is found through a global-sum operation, part of a library of communication subroutine utilities available on most hypercubes. Each processor executes a copy of the global sum and contributes one term to the sum. The answer goes to a prespecified node, which forwards it to the host.

Another common global operation is the global send, which distributes a value from the host or a node to all the other nodes. Such global operations are used so often in hypercube applications they are included in the basic processor-to-processor operations supported by each node's operating system.

Computing π is a relatively simple perfectly-parallel application, although it encompasses several important hypercube fundamentals—such as global operations, the role of the host, and distribution of the task among many nodes. Applications that lend themselves to perfectly-parallel programming include molecular modeling, in which many of a molecule's states are simulated through Monte Carlo techniques. (With Monte Carlo techniques, a large number of possible solutions are generated at random and compared with some preset criteria to determine whether or not they are valid solutions.)

Image recognition is another application for which parallel architectures hold great promise. Here the image of a discrete object must be analyzed and compared with a large number of images of candidate objects. On a hypercube, each of these comparisons can be performed independently and simultaneously, using the same input data in each node.

Moving existing code from a conventional, single-processor machine to a concurrent system has the obvious advantages of preserving an investment in developed, debugged code, as well as the implementation's numerical accuracy and stability. In many perfectly-parallel applications, some portions of the application software remain unchanged from the sequential implementation. The most significant modification is the adding of communications functions that exchange data between nodes. With explicitly-parallel applications, however, the process is more complicated, since segmentation of the application means that boundary conditions must be modified. In some of those applications, rewriting the program so that it runs on a hypercube may be impractical because communications between nodes wind up dominating programming time.

Solving problems with rings and trees

Another benefit of the hypercube architecture is the flexibility of its interconnect scheme. In hypercubes of four or more dimensions, programmers can choose from among several different standard network topologies to match the problem at hand. A programmer can view a four-dimensional network, for example, as a mesh, a tree, or a ring, and direct its internode connections accordingly [see figure]. Each topology is a useful abstraction for writing software for certain types of application.

For example, two-dimensional mesh topologies work well for simulating two-dimensional phenomena, such as the heat flow on a metal plate. Similarly, three-dimensional meshes are good for simulating spatial phenomena, like the dispersion of air pollutants or the diffusion of dopant molecules in a semiconductor. Tree structures are used for global broadcast or combination functions, and for the search algorithms common in artificial-intelligence software.

Rings are useful in situations where one or more data elements must be evaluated against all the others. One such situation is presented by a common astrophysical problem called the n-body simulation of the universe. In this simulation, the gravitational forces on each one of n celestial bodies must be computed with

Distributed memory message passing machines at a glance

Machine	Developer	Year	Topology	Maximum number of nodes	Maximum memory per node, Kbytes	Node CPU	Estimated node performance, megaflops*	Estimated node instruction rate, MIPS†
Waterloop/64	University of Waterloo Ont., Canada	1983	Loop	64	128	8086/87	0.025	0.7
Cosmic Cube	California Institute of Technology (Caltech) Pasadena	1983	Hypercube	64	128	8086/87	0.025	0.7
Mark II	Jet Propulsion Laboratory-Caltech Pasadena	1985	Hypercube	64	256	80286/287	0.035	1.0
iPSC	Intel Scientific Computers Beaverton, Ore.	1985	Hypercube	128	512	80286/287	0.035	1.0
System 14	Ametek Inc. Arcadia, Calif.	1985	Hypercube	256	256	80286/287	0.035	1.0
NCube/ten	NCube Beaverton, Ore.	1986	Hypercube	1024	128	Special[1]	0.3–0.5	2.0
Computing Surface	Meiko Kanagawa, Japan	1986	2-dimensional mesh	84	128	Transputer	—	7.0
iPSC-VX	Intel Scientific Computers Beaverton, Ore.	1986	Hypercube	64	1.5K	Vector [2]	6–20	1.0
T-Series	Floating Point Systems Beaverton, Ore.	1986	Modified hypercube	16384	1.0K	Vector[3]	16–20	7.0
Connection Machine	Thinking Machines Corp. Cambridge, Mass.	1986	Hypercube	65536	0.5	Special[4]	—	0.015
Butterfly	Bolt, Beranek & Newman Cambridge, Mass.	1986	Banyon Switch[5]	256	1.0K	68020/81	0.1	1.0
Mark III	Jet Propulsion Laboratory-Caltech Pasadena	1986	Hypercube	1024	4.0K	Vector[6]	20	1.0

* Megaflops—millions of floating-point operations per second
† MIPS—millions of instructions per second
1. NCube has developed a single-chip node CPU that incorporates a 32-bit processor element, 11 bidirectional communications channels and a memory controller.
2. The Intel VX series uses the iPSC's 80286/80287 node CPU as a communications-control processor in conjunction with a microprogrammed vector-function processor built around Analog Devices 3210/3220 floating-point arithmetic units.
3. The T-Series node uses an Inmos transputer (IMS T414) as a communications-control processor in conjunction with a vector processor of the company's own design, using Weitek 1164/1165 floating-point arithmetic units.
4. The Connection Machine uses a unique 1-bit serial processor, 16 of which are integrated into each physical processor chip.
5. All instructions are broadcast to all nodes, where their execution is synchronized.
6. The Mark III is a joint JPL-Caltech research project; it has a 68020/68081 node CPU and a vector processor of the company's own design, built around Weitek 1164/1165 floating-point arithmetic units.

respect to all the others, for a total of n^2 calculations. Mathematical representations of each body are distributed evenly through the ensemble of processors, which are linked so as to resemble a pearl necklace—the pearls in this analogy being the processors. With one or more bodies allotted to each processor, the forces are computed by passing each body's vital statistics through the ring and around the ensemble. In each processor, gravitational computations are carried out with respect to the bodies allotted to that node. Once a body has made the complete circuit and returned to its original node, all forces imposed on it by the other bodies will have been determined. Another iteration in the simulation can begin when all bodies have passed around the loop.

Several software utilities exist that aid in the writing of parallel programs for these topologies. One utility, for example, simplifies the construction of two-dimensional mesh programs, allowing programmers to define communications for nodes relative to their position in the mesh. The programmer need specify only the direction of communication—up, down, left, or right—and the utility computes the relative address. Researchers are developing more sophisticated utilities for more complex topologies.

Another challenging software topic at the moment is dynamic load balancing. Some kinds of explicitly-parallel applications require the computational load to be continuously balanced while the software is executed. For example, to simulate a fluid vortex, a finely sampled three-dimensional mesh is needed in the area of the vortex center and its radiating arms, but a less finely sampled mesh will do elsewhere. It would be inefficient to simulate the entire area with the accuracy needed for the center and arms. But since center and arms are constantly moving, regions where fine sampling is required move too, which must be accounted for as the program runs.

Several methods have been devised to help with this kind of problem. One is analogous to the physical principle of entropy, where the computational load can be likened to temperature. In this analogy, nodes with heavy computational loads are "hot," those with relatively light loads are "cool." Every few iterations, nodes compare computational loads, and those that are "hot" pass off part of their task to adjacent, "cooler" processors. Future hypercubes may apply similar methods at the operating-system level automatically to redistribute the computational load of a parallel application, without instruction from a human programmer.

While such methods can simplify the programmer's task, they increase the communications needed for an application and degrade overall performance somewhat. Also, incorporating big functions, like load balancing, into the operating system is something of a tradeoff, since the functions slow the system's overall execution.

Maximizing efficiency

While an evenly distributed load is essential for efficient parallel computing, other factors also weigh heavily—such as the amount of time spent on communication between the processors.

A parallel computer's efficiency can be expressed as the ratio of the processing rate of the multinode system to the product of

Prospecting with a hypercube

Seismic simulation is a technique used by the oil industry for evaluating the accuracy of interpretations derived from seismic field data. Since the structure of the earth's substrata in a region cannot be determined with absolute accuracy, oil explorers have to use indirect means.

They combine information from a variety of sources, such as geological data on the prehistoric formation of the region, characteristics of the strata derived from previous oil wells in the region, gravitational measurements, and low-resolution images from seismic soundings of the subsurface. Using the wave equation, a composite picture from these sources can be constructed, and the results are then compared with the field data.

Computers are used to construct the composite that checks the accuracy of the interpretation, based on the field data. Geologists have employed hypercube computers—parallel machines that are gaining acceptance in a variety of scientific and technical fields—for this purpose. The hypercube simulates wave propagation in the earth's crust. The source of the simulated waves is a simple pulse function, which corresponds to an underground dynamite blast or underwater air-gun blast in a field survey. Sensors in the simulation are placed to correspond with the positions of the geophones in the field and pick up simulations of vibrations that would reach the earth's surface.

Layered strata below the surface, with discrete boundaries and faults, present a profile of varying velocities and make such simulations extremely difficult to perform accurately. The wave equation is expressed as a partial differential equation, with varying velocity and density. To solve the equation on a hypercube, the region being simulated must be divided into a grid, where each point has a unique density and velocity coefficient. The program determines the pressure of the acoustic wave at each point in the grid.

The parallel algorithm distributes the task among many processors, dividing the simulated region into subregions, either in strips or squares. For each time step, identical calculations are performed in each processor to compute the pressure for each grid point at each instant. Upon completion of a time step, each processor communicates the results at the edges of its subregions to its adjacent processors, then goes on to the next time step. The simulation continues until the time interval is completed, typically about six seconds of actual (not computer) time. —*P.W.*

the single-node processing rate and the number of nodes in the computer—in other words, the system's actual performance divided by expected performance under the ideal conditions of a direct, linear relationship between nodes and performance. For hypercubes running software amenable to concurrent execution, the ratio is typically in the 80–90 percent range, depending on the type of problem.

Efficiency is determined primarily by two aspects of a parallel algorithm: the ratio of concurrent to sequential calculations, and the ratio of computations to internode communications. The upper limit on efficiency is given by Amdahl's law, first stated in 1967 by Gene Amdahl, then an IBM employee and now a Silicon Valley entrepreneur. Amdahl noted that in any parallel program, the maximum performance gain is equal to the reciprocal of the fraction of the program that is inherently sequential—the part that must be run on a single node. For instance, if a tenth of an application must be run on a single processor, then the best concurrent implementation with more of those processors could theoretically run 10 times faster than the single-processor version.

The goal in designing hypercube hardware is to maximize the number of calculations relative to the node-to-node communications. In general, hypercubes are designed to support a ratio of computational time to communication time of at least 10 to 1, as measured by the ratio of the rate at which the nodes execute instructions to the bandwidth of the communications channels. Programmers need to keep that 10-to-1 ratio in mind when writing hypercube software so as not to overload communications channels.

Breaking into the mainstream: within 5 years?

The trends in hypercube hardware should cause no surprise. The technology is moving toward high-performance, 32-bit node processors, bigger memories on each node (to more than 1 megabyte in some systems), and higher and higher dimensions.

Single-board vector processors, which boost the arithmetic performance of each node in a system, are also finding wider use. Such coprocessors offer high performance in repetitive calculations on ordered sets of numbers, such as matrices and vectors.

Expanding the memory capacity of each node greatly increases efficiency, particularly for medium- to large-scale problems. With more memory, more data can be stored at each node, and more computations performed by the node without communicating results to other nodes. This is particularly important in the current generation of hypercube computers, which use serial communications between nodes. As memory chips cost less and less, nodes with 10s of megabytes will become available. The hypercube architecture is well positioned to take advantage of such VLSI advances.

Technology is moving so fast, it is still unclear whether hypercubes will become the architecure of choice for large-scale parallel computing. But many of the current crop of hypercube machines already support a generalized programming model easily adaptable to other distributed-memory message-passing parallel computers. Whatever interconnect scheme is adopted for future concurrent computers, many applications developed today will be compatible.

A growing number of users are developing software for future generations of machines vastly more powerful than today's supercomputers. Hypercube applications have been proposed in such fields as molecular modeling for drug design, computer-aided design of electronic and mechanical systems, structural analysis, oil-reservoir simulation, and fluid dynamics.

As might be expected with such a young computer technology, many software developments are still on the horizon. Research into automated, dynamic methods of distributing and balancing applications has only just begun. Such methods are likely to be crucial in bringing the hypercube architecture into the mainstream of computer applications.

To probe further

The construction in 1983 of the first hypercube computer, at California Institute of Technology, Pasadena, was described by Charles L. Seitz in "The Cosmic Cube," *Communications of the ACM*, January, 1985. The article has become a standard reference on hypercube architecture. At the annual International Conference on Parallel Processing, experts exchange ideas on hypercubes and other parallel architectures. For copies of the proceedings of past conferences, contact the IEEE Computer Society, P.O. Box 80452, Worldway Postal Center, Los Angeles, Calif. 90080. Other material on hypercubes includes "Minis and mainframes," by Paul Wallich and Glenn Zorpette (commentary by C. Gordon Bell), [*Spectrum*, January 1986, p. 36].

About the author

Paul Wiley (M) is hardware-product manager at Intel Scientific Computers, Beaverton, Ore., where he is developing fast arithmetic hardware for the company's iPSC family of hypercube systems. He holds a BSEE from Virgina Polytechnic Institute and State University in Blacksburg, and an MSEE from California State University, Fullerton. ◆