A PennWell Publication

COMPUTER DESIGN

SEPTEMBER 1, 1985

AR-416

LOCAL AND GLOBAL NETWORKS

LOGIC ANALYZERS OFFER NEW CHOICES IN PERFORMANCE AND PRICE

DUAL-POINTER FIFO ELIMINATES BUS ARBITRATION

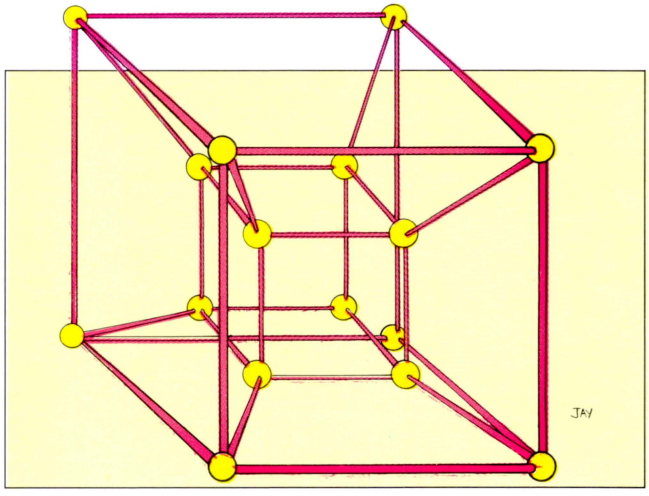
SPECIFY WISELY TO LOWER POWER SUPPLY COSTS

Order Number: 280197-001

CONCURRENT COMPUTERS IDEAL FOR INHERENTLY PARALLEL PROBLEMS

Processors and nodes mimic scientific problem geometry with expandable, parallel processing architecture.

by Ray Asbury,
Steven G. Frison, and
Thomas Roth



Concurrent processing represents the most promising long-term approach to achieving affordable, accessible supercomputing now that the limitations of supercomputers—especially those handling combined scalar and vector operations—are becoming evident. Concurrency is a high-level, or global, form of parallelism that denotes independent operation of a collection of simultaneous computing activities, rather than the lockstep connotation of the more familiar term "parallelism." Concurrency is essentially an interactive parallelism that allows asynchronous operation of processors in a multiprocessor system.

Solving problems on a concurrent machine requires partitioning them into a number of segments that can run independently on more than one processor. Many applications, particularly in scientific computing, lend themselves to this partitioning. Intel's parallel architecture computer, the hypercube, is well suited for these partitioned applications.

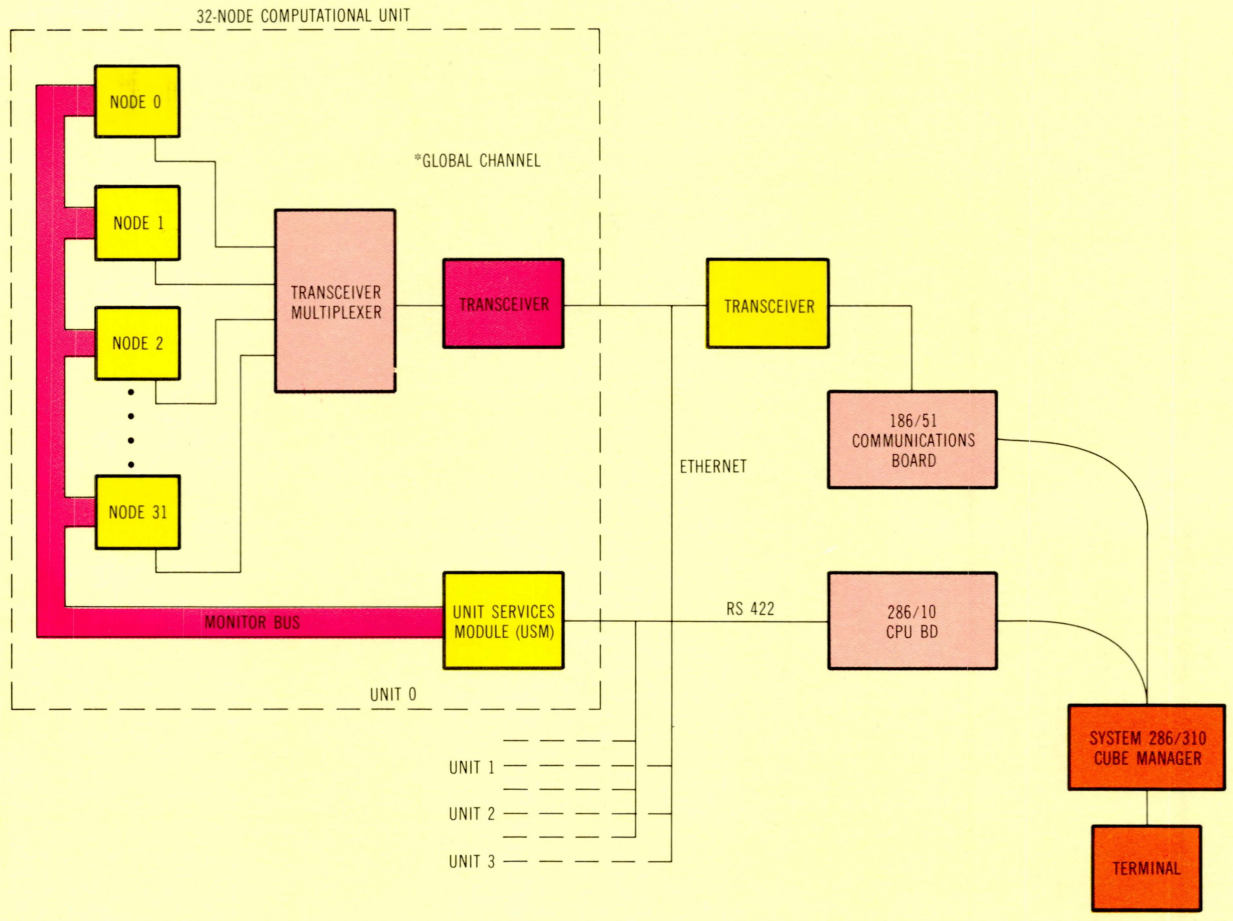
The iPSC hypercube achieves concurrency through an ensemble of loosely coupled, independent processors executing portions of a larger computational problem simultaneously. The hypercube consists of 32, 64, or 128 microcomputers connected to each other via point-to-point communications channels. Each processor is connected directly to a local host processor—the cube manager—via a global communications channel. The cube manager supports the programming environment and serves as the cube's system manager.

Hypercube architecture originated from research at the California Institute of Technology sponsored by the Defense Advanced Research Projects Agency (DARPA). This architecture uses individual microprocessors and associated local memories to form computational nodes that are tied together in a network. For a hypercube of dimension n, 2^n nodes are

Ray Asbury is an applications specialist at Intel Corp (Beaverton, OR). He holds an MS in physics from the University of New Mexico.

Steve Frison is system software manager at Intel Corp. He holds a BS in mathematics and computer science from Portland State University.

Tom Roth is a senior systems engineer at Intel Corp. He holds a BS in applied science and engineering from Portland State University.



Sporting a cube-managing 286/310 microcomputer and up to four 32-node computational units connected by global and diagnostics channels, the iPSC computer systems depend on the Xenix operating system for computer management, and Fortran and C for most of its application programs. An Ethernet network allows local communications.

interconnected by point-to-point communications channels. Hypercube architecture calls for individual nodes to operate independently on a subsection of a larger problem. These individual nodes work independently, according to resident process instructions, on data resident in a specific processor's memory. This data can come through a message-passing system from processes resident in other nodes, or from the cube manager. Process code is written in ordinary sequential languages. Fortran, for example, is used with operating system primitives provided by the cube's operating system. These primitives allow programmers to direct message sending and receiving.

Why hypercube?

The hypercube topology presents several advantages. First, for large systems consisting of hundreds, or even thousands of nodes, the scale of the architecture can be increased by expanding the dimensionality or size, of the cube. Second, its distributed memory (the memory resident at each node) avoids the problems of contention between many processors for a shared memory. Third, its communication properties of high data bandwidth (which grows as $N\log_2 N$), and low message latency (the worst case being $\log_2 N$) help hypercube-based systems achieve computational efficiency. Finally, the robust interconnect structure makes the hypercube adaptable to other topologies, such as ring, tree, and two- and three-dimensional meshes.

Application programs are developed and compiled on the cube manager and then downloaded to the cube nodes. The cube manager is an Intel 286/310 multi-user supermicrocomputer. This supermicro

runs the Xenix 3.0 operating system, which is Intel's version of Unix, as well as associated Fortran and C compilers.

Unlike many parallel processing architectures, the iPSC is a multiple-instruction, multiple-data (MIMD) stream machine that uses message passing rather than shared storage for communication between nodes. Message passing is used to perform the basic unit of cube computation—the "process." A process is defined as a sequential program including system calls that cause messages to be sent and received. In fact, processes communicate only by sending and receiving messages. This architecture provides increased overhead efficiency compared to memory-sharing schemes.

Computing via multiple processes

A single node may contain many processes that perform computations. Computations are distributed through the computer and executed concurrently, either by virtue of being in different physical nodes, or by being interleaved in execution within a single node.

The hardware model of the iPSC is quite similar to the process model of computation. As a result, instead of formulating a problem to fit on nodes, and on the physical communication channels that exist only between certain pairs of nodes, the software developer may formulate a problem in terms of processes and virtual communication channels that connect all processes. This abstraction requires the cube message system to route messages efficiently from any one process to any other process.

The code for a process may be written in any combination of Fortran, C, or ASM286, the 80286's assembly language. It is linked to the node application library using the 80286's binder utility (BND286). Finally, the user process is bound to the kernel with a utility (BLD286) to create a loadable object module.

Each process has a unique 32-bit identifier in which two 16-bit fields are defined. These are the processor (node) number and the process number (process ID) within the node. Process IDs in the range 32,768 to 65,535 are reserved for the kernel. If the process number field is treated as a signed number, user process IDs are positive and reserved process IDs are negative. The size of available memory within the node limits the number of processes per node. The node number ranges from zero to the number of nodes in the cube minus one. In a 32-node cube, for example, the node numbers range from 0 to 31.

The node kernel provides basic services, such as interprocess communication, process management, physical memory management, and protected address space. In addition, the node kernel forms a foundation for other operating system functions.

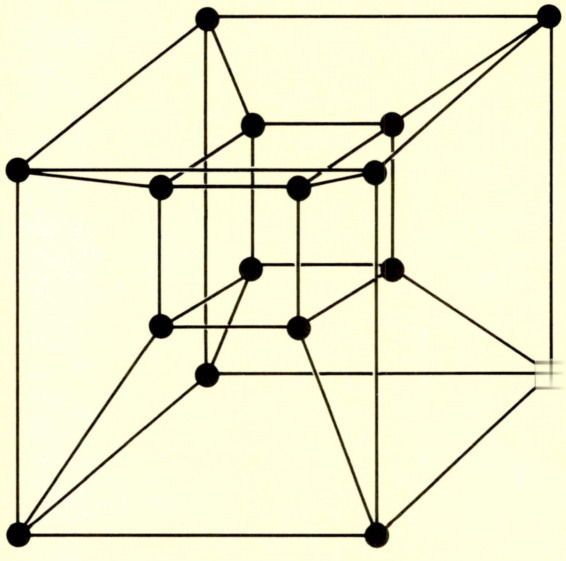
Next, a copy of the node kernel and lined application processes is loaded into each node after initialization and confidence testing have been successfully completed.

Node communication

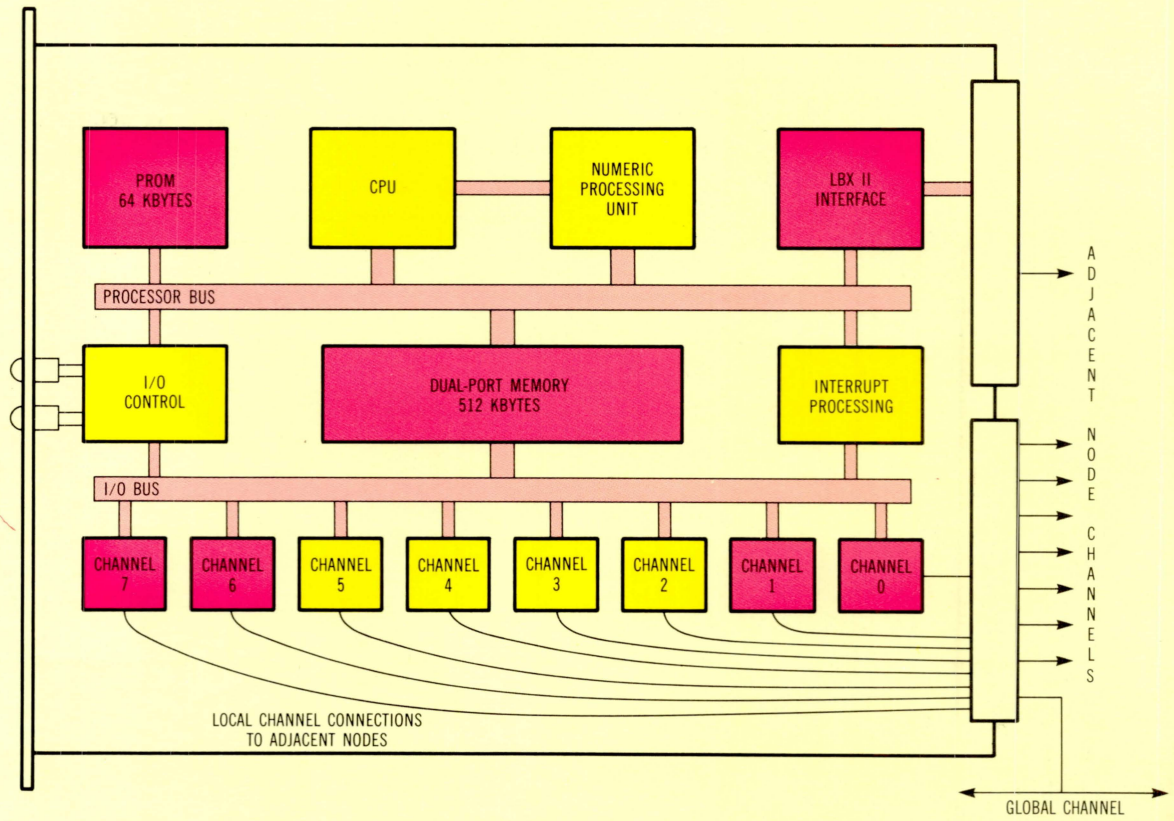
The node operating system provides user processes with a flexible set of communication primitives. The communication mechanisms are the same whether the processes are in the same node, mother nodes, or in the cube manager. The node operating system, in conjunction with the 80286, also provides protection for processes executing within the nodes. This design prevents errors in processes from being transferred to other processes.

The node operating system uses a common message format to support message passing functions. One possible message format is for messages to be passed "by value." This means that a copy of the message from the sender to the receiver is always made, even if the processes are on the same node. Messages may be queued in transit, but message order is preserved between any pair of processes. Message routing in the cube is entirely a function of the node operating system. Shared memory is never used as a communication mechanism.

Interprocess communication is performed through "channels." A channel is a system construct used



A hypercube (binary cube or Boolean n-cube) connects $N = 2^n$ microcomputers (nodes) through point-to-point communication channels. A 16-node ($n = 4$) cube has each node physically connected to its four nearest neighbors.



There are $\log_2 N$ channels per node. An actual iPSC node fits on a 9- by 11-in. Eurocard and contains a self-sufficient microcomputer.

to manage a process's message requests. There is no synchronization between processes when a channel is established. One process communicates with another process simply by opening up the channel and initiating send and receive requests.

Because message passing between processes is asynchronous, there is no guarantee that a process has made a receive request before the node receives the message. To ensure proper message reception, messages that are received by a node prior to a receive request are buffered by the node operating system until the request is made. The message is then transferred into the process message buffer.

A process performs message sending and receiving by invoking system calls. Send and receive merely enable a message transmission or reception. If the action is not satisfied, a request is created that remains pending until it is satisfied. Program execution may continue concurrently with one or more pending communication requests.

Once a channel has been opened, a send and its corresponding receive may occur in either order. It is generally most efficient if the receive is executed before a message arrives at a node. The receive is, therefore, delivered to the process rapidly, without occupying a node operating-system buffer for an extended period.

A programmer can construct the application to incorporated checkpoints, if system reliability demands it. Additional flexibility is possible because a single message may often contain up to 16 kbytes. Messages larger than 1 kbyte are automatically fragmented and reassembled at the destination node in a process transparent to the user.

Point-to-point communications between adjacent nodes perform the bulk of message transfers. Moreover, reliable message delivery is guaranteed between a node and its nearest neighbors. To reduce communications overhead, end-to-end message acknowledgement is not used between a node and more distant nodes (non-neighbors). This feature can be provided at the applications level, if needed.

Concurrent or vector?

When an application is to be programmed for the hypercube, algorithms must be developed to define the discrete component processes of real physical problems. The process descriptions are then topologically assigned to nodes by the programmer to model events as closely as possible. Many types of

concurrent applications can be mapped to the cube. These applications include geophysics, astrophysics, network simulation, image processing, statistics, or any research where the problems exhibit concurrency.

The program to solve a problem that is inherently concurrent in nature can be separated into two components. First, there is a scalar portion, called the loop, or problem space, which mathematically describes the problem boundary. Then there is a vector portion (inner loop), which is a software model of the interaction between the individual problem elements. A common problem in molecular dynamics—the diffusion of one group of molecules through a medium—is a good example of how concurrent processing and vector processing differ in the way they handle similar problems.

Scalar-bound solutions

On a vector or array processor, the parallelism inherent in the mathematical model of the diffusing molecules (the vector portion of the problem) facilitates high performance. The scalar portion of the problem, however, which describes the interaction of the molecules with the vessel walls, limits performance, because it cannot be vectorized and must run sequentially. Because the scalar portion also limits an optimal solution of the problem, the very best vector performance will have little effect on increasing the overall performance of the application beyond this point. In effect, the solution of the problem becomes scalar-bound.

A concurrent machine, on the other hand, shares the load equally over a large number of processors. Each processor solves a small portion of the overall task and interacts over high-speed communication channels. This division of labor achieves the concurrency needed for the task.

The task of any processor in the hypercube system is scaled proportionately to the size of the segment being processed. Each processor solves a portion of the outer loop and a portion of the problem's inner loop. Both vector and scalar performance remain proportional because the hypercube architecture mimics the structure of the problem itself.

A concurrent machine such as the hypercube can also handle non-numerical problems, such as event-driven simulation and artificial intelligence. This capability is not found in conventional supercomputers, because their architectures are optimized for handling vector calculations.

Commercially available components were used to conceive this generation of concurrent computers. These computers perform at up to one third the speed typical of a Cray-1. For example, the iPSC family incorporates nodes based on the 80286 microcomputer, the 80287 numeric coprocessor, and the 82586 LAN coprocessor. The 82586 LAN coprocessors handle message passing between processing elements.

Each node contains 512 kbytes of dynamic RAM, and 64 kbytes of ROM. The ROM contains the node monitor, which is responsible for node initialization at power-up or system reset. The monitor verifies that the node is operable, and loads the kernel and application software. The monitor initializes the system by resetting and enabling the node memory, communication controllers, I/O controller, interrupt controller, and CPU. The monitor is also responsible for node confidence testing. The confidence test verifies the node's printed circuit board by running a node confidence test (NCT) on the RAM, peripheral devices and communications controllers.

Each of the three hypercube systems within the iPSC family includes a cube that contains the network of microcomputer nodes, and a System 286/310 microcomputer for overall system management. The cubes for the three systems within the family include one, two, or four 32-node computational units.

In addition to seven I/O channels for internode communication, every node board has an eighth channel—an Ethernet link known as the global channel—for interface with the cube manager. For memory expansion or extended processing capacity, the node processor's local memory bus on each board ties into the iLBX-II bus interface, which is defined in the Multibus II standard.

Selecting components

While the hypercube architecture makes the construction of powerful computers possible using off-the-shelf VLSI devices, certain considerations govern the actual component selection. Although programming concurrent processors is only slightly more difficult than programming a sequential CPU, memory security at each node is critical to guarantee reliable operation. Local memory, for example, must accommodate process instructions as well as high-speed message buffering. In the iPSC, each node carries in local memory a copy of the operating system that directs message passing.

The 80286 microprocessor is a good choice for the iPSC for several reasons. It has built-in memory protection, and supports the isolation of the operating system and tasks, as well as program and data privacy within tasks. In application, local descriptor tables for each process resident at a given node provide a partition between the operating system kernel and user space, and between multiple-user process spaces. The 80286 also has a 16-Mbyte real address space which is more than adequate for first-generation hypercubes. A companion communications controller and numeric coprocessor are available to support the 80286. Finally, it supports a variety of

Wait a MOMENT

An example of Fortran programs that use the hypercube concurrent processing capabilities are the programs (processes) STATS and MOMENT. Running on a one-dimensional hypercube, STATS computes the mean and standard deviation of lists of floating-point numbers on one node.

MOMENT runs concurrently on two nodes, and is used by STATS to do most of its arithmetic concurrently; MOMENT and STATS run concurrently on one node. MOMENT repeatedly receives lists of floating-point numbers, computes a sum of powers of those numbers, and sends the sum back to the host process. The power is equal to the node id plus 1, so node 0 computes the sum of the numbers and node 1 computes the sum of the squares of the same numbers.

Data is passed between the processes using CALL SENDW, CALL RECVW, CALL SENDMSG and CALL RECVMSG. When the process STATS needs to do arithmetic, it uses CALL SENDMSG to call MOMENT, and then passes the information to MOMENT. MOMENT is simultaneously awaiting data from STATs, using CALL RECVW.

Once MOMENT processes the data, it issues a CALL SENDW and waits for STATS to issue a CALL RECVMSG. At this point, the program sends the data back to the host process (STATS) on the originating node.

```
                PROGRAM STATS
      INTEGER K,M,N,CID,HOST,BYTES,XLEN,TYPE
      INTEGER COPEN
      DOUBLE PRECISION X(100),SUM,SUMSQS,TEMP,MEAN,STDEV
      DATA XLEN /800/, TYPE /1/, HOST /-1/
C
C     Open a channel. Use a process id equal to the node id.
C
      CID = COPEN(HOST)
C     Start the main loop. Read the data from the terminal.

   10 WRITE (*,*) 'Enter n, x(1), ..., x(n)'
      READ(*,*) N, (X(I),I=1,N)
      IF (N .LE. 0) STOP
      WRITE(*,*) 'N = ',N
C
C     Ask node 0 to compute sum of x(i) and
C     ask node 1 to compute sum of x(i)**2.
C
      BYTES = 8*N
      CALL SENDMSG (CID,TYPE,X,BYTES,0,0)
      CALL SENDMSG (CID,TYPE,X,BYTES,1,1)
C
C     Wait for two replies, which can come in either order.
C     M is the id of the node originating the reply.
C
      BYTES = 8
      DO 40 K = 1, 2
        CALL RECVMSG (CID,TYPE,TEMP,BYTES,BYTES,M,M)
        IF (M .EQ. 0) SUM = TEMP
        IF (M .EQ. 1) SUMSQS = TEMP
   40 CONTINUE
C     Compute the statistics and print the results.

      MEAN = SUM/N
      STDEV = DSQRT (SUMSQS/N - MEAN**2)
      WRITE(*,*) 'MEAN = ', MEAN
      WRITE(*,*) 'STDEV = ', STDEV
      WRITE(*,*)
C
C     Do it again
C
      GO TO 10
      END
```

```
                PROGRAM MOMENT

      INTEGER I,M,N,CID,HOST,BYTES,XLEN,TYPE
      INTEGER MYNODE,COPEN
      DOUBLE PRECISION X(100),SUM
      DATA XLEN /800/, TYPE /1/
C
C     Find the node id
C
      M = MYID ()
C
C     Open a channel. Use a process id equal to the node id.
C
      CID = COPEN(M)
C
C     Start the main loop.
C
   10 CONTINUE
C
C     Wait for n and x.
C
      CALL RECVW (CID,TYPE,X,XLEN,BYTES,HOST,HOST)
      N = BYTES/8
C
C     Compute the desired sum.
C     Note that the node id is involved in the arithmetic.
C
      SUM = 0.0D0
      DO 20 I = 1, N
        SUM = SUM + X(I)**(M+1)
   20 CONTINUE
C
C     Send the sum back to the host.
C
      CALL SENDW (CID,TYPE,SUM,8,HOST,HOST)
C
C     End of the main loop.
C
      GO TO 10
      END
```

standard software environments and the CPU's local bus interface boasts an 8-Mbyte/s bandwidth.

The 80287 numeric processor handles 32-bit, 64-bit, and 80-bit floating-point arithmetic operations in conformance with the draft IEEE-754 floating-point standard. Each 82586 LAN coprocessor provides a 10-Mbit/s pathway for internode transfers. Typically, the devices permit a 100-MIPS aggregate communication rate for the 128-node system. The complex responsibilities of each node dictates selection of devices that integrate the maximum number of functions, yet interact efficiently (a 128-node

hypercube incorporates eight separate communications channels/node, for a total of 1024 active I/O ports).

Without VLSI that integrates most of the control functions for at least one port, the concurrent architecture would become cost-prohibitive. With nodes being formed from such VLSI-intensive devices, compact hypercube systems can be easily expanded. The iPSC system is expanded from one to two or four 32-node computational units by connecting them with cables.

Future of concurrent processing

Concurrent processing allows great increases in computational power. Of all concurrent processing architectures, the hypercube's primary benefit is its scalability. Systems made up of thousands of nodes may become typical.

In the near future, the majority of hypercube system enhancements will focus on greater performance. These performance enhancements can occur in many areas. For example, 32-bit architectures, higher performance processors, and the addition of more floating-point capabilities are all possible. These can more than double individual node board performance. Also, increased memory capabilities will allow larger granularity per node, permitting the solution of larger problems.

As levels of computing capability increase, more processors in larger systems may become the norm, moving beyond the 128-node level. In the more distant future, implementations will use dedicated VLSI devices, rather than off-the-shelf components. And rather than printed circuit boards, entire nodes—possibly multiple nodes—will be contained on single chips. Future machines may also use wafer-scale integration techniques, leading to a complete machine-on-a-chip.

INTEL CORPORATION, 3065 Bowers Ave., Santa Clara, CA 95051; Tel. (408) 987-8080
INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000
INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

Printed in U.S.A./SC5-090/3K/CD/MB
ISC