Parallel Computing 17 (1991) 1285–1302
North-Holland

Performance of the Intel iPSC/860 and Ncube 6400 hypercubes *

T.H. Dunigan

Mathematical Sciences Section, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Received March 1991

Abstract

Dunigan, T.H., Performance of the Intel iPSC/860 and Ncube 6400 hypercubes, Parallel Computing 17 (1991) 1285–1302.

The performance of the Intel iPSC/860 hypercube and the Ncube 6400 hypercube are compared with earlier hypercubes from Intel and Ncube. Computation and communication performance for a number of low-level benchmarks are presented for the Intel iPSC/1, iPSC/2, and iPSC/860 and for the Ncube 3200 and 6400. File I/O performance of the iPSC/860 and Ncube 6400 are compared.

Keywords. Hypercubes; Intel iPSC/860; Ncube 6400; performance results; computation benchmarks; communication benchmarks.

1. Overview

1.1. Introduction

This report compares the results of a set of benchmarks run on the Intel i860-based hypercube, the iPSC/860, and the Ncube 6400 with earlier results reported in [4] for older Intel hypercubes (iPSC/1 and iPSC/2) and the first generation Ncube hypercube (Ncube 3200). These hypercubes are descendants of the pioneering work done at Caltech [14]. A hypercube parallel processor is an ensemble of small computers interconnected by a communication network with the topology of an n-dimensional hypercube. Each processor, or node, has its own local memory and communication channels to n other nodes. The processors work concurrently on an application and coordinate their computation by passing messages.

We are interested in the performance of hypercubes for several reasons. First, our main area of research is the development of algorithms for matrix computations on parallel computer architectures. To produce algorithms that make effective use of a parallel architecture it is necessary to understand the basic structure of the architecture and the relative performance and capacities of the fundamental components – CPU, memory, communication (message passing), and I/O. Second, some of our development work is done on hypercube simulators, both to debug and to analyze our algorithms [5]. Performance results from real hypercubes enable us to construct more accurate simulators. Finally, a set of benchmarks and performance results can help us evaluate new implementations or architectures.

* The work was supported by the Applied Mathematical Sciences subprogram of the office of Energy Research, US Department of Energy.

Table 1
Hypercube configurations used in tests

	Configurations for tests				
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200
Number of nodes	128	64	64	64	64
Node CPU	i860	80286/287	80386/387	32-bit	32-bit
Clock rate	40 MHz	8 MHz	16 MHz	20 MHz	8 MHz
Memory/node	8M	512K	4M	4M	512K
Nominal data rate	22 Mbps	10 Mbps	22 Mbps	20 Mbps	8 Mbps
Node OS	NX v3.2	v3.0	NX v2.2	Vertex 2.0	Vertex v2.3
C compiler	PG v2	Xenix 3.4	C-386 1.8.3A	xucc v2.0	CF&G v1.0

In the remainder of this section, we summarize the hypercube configurations and programs used in our test suite. Section 2 discusses the hypercube architectures in more detail, emphasizing the distinctive features of each implementation. The computational power and memory capacity of the hypercubes and their message-passing performance are compared in sections 3 and 4, respectively. Section 5 examines the performance of the file system (I/O), including the concurrent file system (CFS) of the iPSC/860 hypercube. Section 6 summarizes and reports aggregate performance.

1.2. Test environment

Five commercially available 64-node hypercubes were used for our benchmark suite. We have both Intel and Ncube 3200 hypercubes at Oak Ridge National Laboratory and have access to a 64-node Ncube 6400 at Ames National Laboratory. The configurations utilized in the tests are summarized in *Table 1*. In this report, 'iPSC/1' refers to the first generation Intel hypercube, 'iPSC/2' refers to the second generation Intel hypercube, and 'iPSC/860' to the new i860-based Intel hypercube. 'N6400' is used to denote the new Ncube hypercube, and 'N3200' refers to the first-generation Ncube.

The test programs were written in C (except where noted) and were run on the iPSC/1 hypercube in the first quarter of 1987. Tests of the iPSC/2 and Ncube 3200 hypercubes were performed in the second quarter of 1988 and revised in the last quarter of 1990. The iPSC/860 and Ncube 6400 hypercubes were tested in the first quarter of 1991. The large model memory option was used with the C compiler for the Intel iPSC/1 (*-Alfu*), and stack checking was disabled for the iPSC/1 and Ncube 3200 C compilers. An optimization level of *-O2* was used for C compilers for iPSC/860 and Ncube 6400. The test suite was selected for simplicity of

Table 2
Benchmark programs used in tests

	Benchmark summary
Caltech	integer and floating point arithmetic operations + − */
Sieve	finding primes using integer arithmetic
Floatmath	double precision floating point arithmetic
Dhrystone	integer arithmetic and functions
Whetstone	double precision floating point arithmetic and built-in functions
Malloc	free memory test using 1 K malloc
Ring	Gray-code ring message passing
Echo	message echo
Spincom	*N* iterations of a loop timed with simultaneous message routing

implementation and widespread use, permitting us to implement the tests with few source changes and to compare the results to other architectures reported in the literature. For the computation tests, the call to the node clock subroutine and the code to send the result back to the host were the only source-code changes made in porting the tests from one vendor to another. *Table 2* summarizes the test programs.

2. Configurations

Each hypercube configuration consists of a hypercube attached to a host processor. The host processor is used for program development and as an interface to the outside world for the hypercube. A typical hypercube application program consists of one or more node programs and usually a host program to provide input data and report results.

2.1. Intel

The first generation Intel hypercube, iPSC/1, consists of from 32 to 128 nodes attached to an Intel 310 host processor. The host and node processors are 80286/80287 running at 8 MHz. Each node has 512 Kb of main memory and is attached to the host via a global communication channel. The iPSC/1 can be expanded to 4.5 Mb per node, and a vector processor option is available as well. The host operating system is Xenix and supports the typical UNIX program development environment. Since the host and node CPUs are the same, one compiler supports both environments. Fortran and C are supported on the hypercube, and Lisp is supported with the large memory option. The iPSC/1 hypercube is a single-user subsystem.

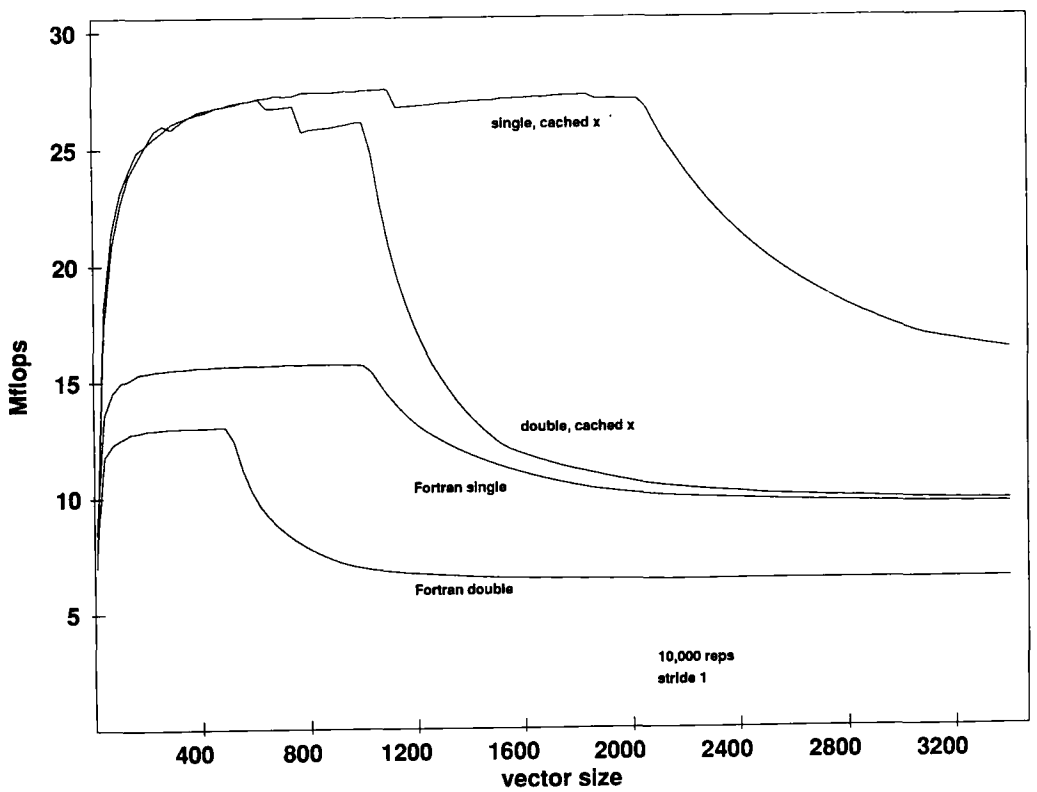

Fig. 1. i860 Fortran and assembler dot product performance.

The node operating system supports message routing, asynchronous communications, and multi-tasking within each node. A node-to-host logging facility is provided for application debugging and diagnostics. Messages larger than 1024 bytes are broken into 1024-byte segments. A node debugger is provided on the host as well as a simulator.

The second generation Intel hypercube, iPSC/2, consists of from 32 to 128 nodes attached at an Intel 301 host processor. The host and node processors are 80386/80387 processors running at 16 MHz, were each node processor has a 64 Kb cache memory. Each node has 4 Mb of main memory, expandable to 16 Mb. Node 0 is attached to the host processor. The host processor runs System V UNIX. Subcube allocation is supported, allowing multiple users to access the hypercube. A debugger is also provided, and a vector processor option is available.

Node communication is supported by direct-connect routing modules on each node. Messages of 100 bytes or less travel as part of the route-acquisition protocol. The node operating system supports multi-tasking, asynchronous communication, and remote I/O support to the host system.

The iPSC/860 hypercube utilizes the same communication hardware as the iPSC/2, but a 40 MHz i860 RISC processor replaces the 80386/80387. The i860 has an 8 Kb data cache, 8 Mb of main memory, and multiple arithmetic units, permitting multiple operations per cycle. The pipe-lined floating point units are capable of a combined peak rate of 80 megaflops (32-bit) or 60 megaflops (64-bit). The present compilers are only achieving 15 megaflops, and even hand-coded assembler routines may achieve only two-thirds of peak performance. Peak performance is difficult to achieve because of various memory delays (cache miss, page-translation miss, DRAM access delays, etc.). *Figure 1* compares the performance of a dot product in single and double precision for both Fotran and assembler.

2.2. Ncube

The Ncube 3200 hypercube consists of from 4 to 1024 nodes attached to an 80286/80287 host. The node processor is a 32-bit chip that was designed by Ncube and runs at 8 MHz. The chip contains both floating point and 10 communication channels. It is surrounded by 512 Kb of memory. The processor chip is also used as the interface processor between the hypercube and the host. The custom chip permits a large number of processors in a small form factor. For example, a four node board is available for the IBM PC/AT. The hypercube may be divided into logical subcubes for multi-user use [12].

The host operating system is 'UNIX-like' but still lacks many of the features of a mature UNIX environment. Both C and Fortran compilers are provided along with a node-level debugger. The node operating system supports message routing and asynchronous communication.

The Ncube 6400 is a 64-bit chip driven by an 80 MHz crystal supporting from 1 to 64 Mb of memory. The chip contains a networking communication unit that supports 13 channels, cut-through routing, and broadcast [11]. The host processor for our tests was a Sun 4 system. Cross-compilers and linkers are provided on the Sun, along with utilities to initialize, load, and debug the hypercube.

3. Computation benchmarks

3.1. Arithmetic tests

To compare our test results with earlier hypercube benchmarks performed at Caltech [9], we implemented a series of tests to measure the arithmetic speeds of the CPU for integer and

Table 3
Arithmetic operation times (microseconds)

	Arithmetic times microseconds					
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200	VAX
INTEGER*2+	0.1	2.5	1.1	0.6	4.5	3.3
INTEGER*4+	0.1	5.0	0.6	0.5	4.9	1.8
INTEGER*2*	0.3	4.0	1.3	0.8	6.0	5.1
INTEGER*4*	0.3	36.5	1.5	0.8	6.3	2.4
REAL*4+	0.1	38.0	5.5	0.8	16.6	7.1
REAL*8+	0.1	41.5	6.6	1.3	11.5	4.6
REAL*4*	0.1	39.5	5.9	0.8	18.5	9.3
REAL*8*	0.1	43.0	7.0	1.5	13.5	6.5
REAL*4* + * + *	0.18	23.1	3.4	0.4	10.6	5.6
REAL*8* + * + *	0.05	24.1	3.8	0.7	7.8	4.4

floating point arithmetic. The time to perform a binary arithmetic operation and assignment in a loop was measured for both single and double precision scalars in C. The time for the loop overhead was subtracted, and the resulting time divided by the number of iterations to give a rough estimate of time-per-operation. *Table 3* shows the results of those tests. In the table, Fortran notation is used for clarity to describe the data types; the tests were run in C.

For purposes of comparison, times for a DEC VAX 11/780 with FPA and running UNIX 4.3 bsd are included. The times illustrate both CPU speed and compiler differences. The last two rows give the average operation time for a sum of three products. Such an expression permits the arithmetic units to retain intermediate results and get improved performance. It should also be noted that C requires that all floating point expressions be calculated in double precision and that all integer expressions be calculated in the world size of the machine. The default integer word size is 16 bits for the iPSC/1 and is 32 bits for the others. The degree to which the compilers comply to the C requirement varies. For double precision floating point computations, an Ncube 3200 node is roughly three times faster than an iPSC/1 node, operating at 0.12 megaflops to the 80287's 0.04 megaflops. An 80387-based iPSC/2 node operates at 0.29 megaflops. An i860 node is measured at 18 megaflops compared to 2.5 megaflops for the Ncube 6400.

3.2. Synthetic tests

The results from the arithmetic operation tests are consistent with the next level of tests performed using a simple integer test of finding primes (*sieve*) and a sequence of dependent floating point operations (*floatmath*). The times for 100 iterations of finding the primes from 1 to 8190 and for 256000 repetitions of the double precision floating point arithmetic operations are listed in *Table 4*. In the *sieve*, variables of type *register int* are used, which means 16-bit arithmetic for the iPSC/1 and 32-bit arithmetic for the others.

Table 4 also shows the times for 50,000 iterations of the Dhrystone test. The test exercises integer arithmetic, function calls, subscripting, pointers, character handling, and various conditionals [15]. There are no floating point calculations. The times are from tests using the *register* storage class of C. The test uses the type *int*, which means 16-bit arithmetic for the iPSC/1 C compiler and 32-bit arithmetic for the others. The table also compares times for one million Whetstone operations. The Whetstone test measures double precision floating point performance, conditionals, integer arithmetic, built-in arithmetic functions, subscripting, and function calls [2]. The iPSC/860 is three to thirteen times faster than the Ncube 6400 for the computational kernels in *Table 4*.

Table 4
Execution time in seconds for various test suites

	Simple computation tests seconds					
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200	VAX
Sieve	0.06	29.3	5.7	0.41	21.4	13.6
Dhrystone	0.53	55.9	6.2	7.3	43.7	30.0
Floatmath	0.41	66.3	11.6	1.3	15.2	10.3
Whetstone	0.12	5.6	0.8	0.4	2.5	2.2

Table 5
Node memory capacity and usage

	Memory capacity per node Kilobytes				
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200
Total	8192	512	4096	4096	512
Available	7299	366	3717	3831	453

The double-precision version of LINPACK [DON91] and the Livermore Loops were used to compare the performance of Fortran. Using the *pgf77* v2.0 with the -O3 option, the iPSC/860 had a LINPACK rating of 6.0 megaflops and a harmonic mean of 5.7 megaflops on the Livermore Loops. The Ncube 6400 had a LINPACK rating of 0.5 megaflops and a Livermore harmonic mean of 0.6 megaflops using the -O2 option.

3.3. Memory utilization

The amount of memory available to an application on a node was measured using the *malloc*( ) function of C. The test program requested memory in kilobyte increments. *Table 5* shows the amount of memory available to the application program compared to the total amount of physical memory for the test configuration.

The difference between the total and available memory gives a rough measure of the amount of memory required by the node for its operating system, message buffers, and C run-time environment. For the 80286 architectures, memory is managed in 64 Kb segments, so there may be additional small chunks of free memory available.

For any computer system, the amount of main memory is a critical metric, and there never seems to be enough. For a hypercube, the amount of node memory can determine the size of problem that might be solved. Shortage of memory is paid for in problem-solution time (due to the I/O or message-passing delays) and in programmer time (due to the additional coding required to multiplex the node memory).

4. Communication benchmarks

4.1. Echo tests

To measure communication data rates, an echo test was constructed. A test node sends a message to an echo node. The echo node receives the message and sends it back to the test

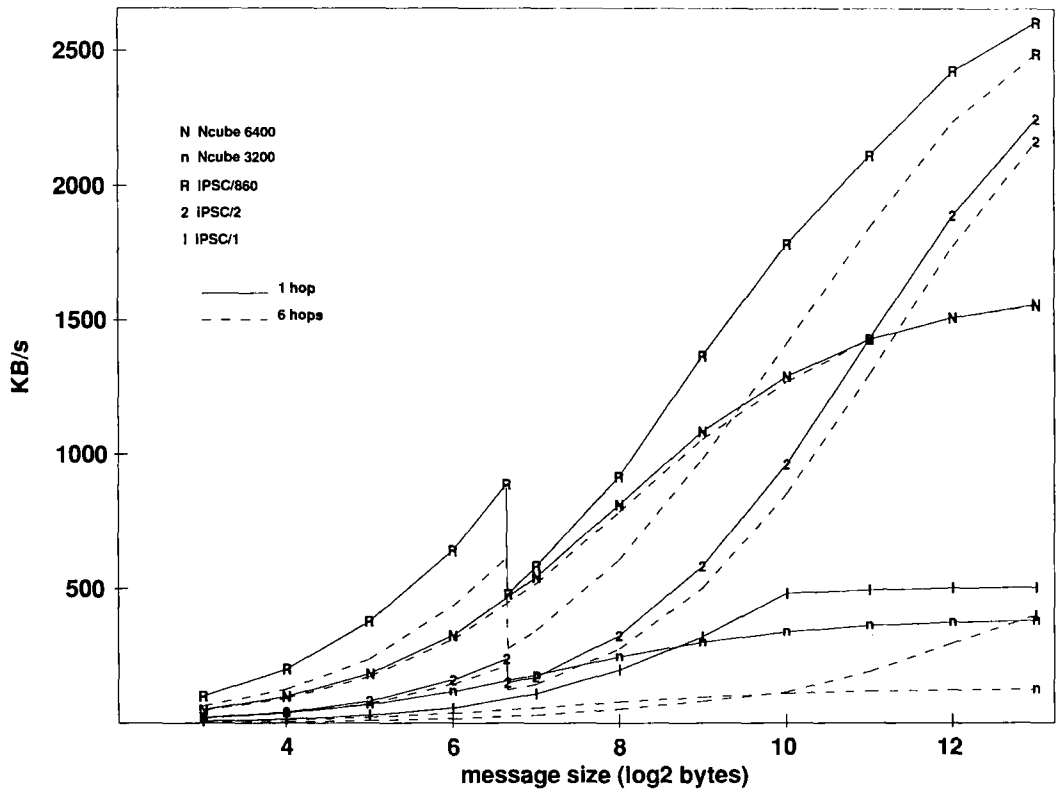

Fig. 2. One-hop data rates.

node. The test node measures the time to send and receive the message N times. *Sendw / recvw* were used on the Intel iPSC/1, *csend* and *crecv* were used on the iPSC/860 and iPSC/2, and *nwrite / nread* were used on the Ncube. *Figure 2* shows the data rates averaged over 100 repetitions for the five hypercubes using various message sizes, where the echoing node is one hop away. For a message size of 8192 bytes, the hypercubes had the following utilization of the peak hardware bandwidth. The Ncube 3200 has a peak data rate of about 380 Kb/s or about 38% of its bandwidth. The Ncube 6400 has a peak data rate of about 1558 Kb/s or about 62% of its bandwidth. The iPSC/1 has a peak data rate of about 505 Kb/s or about 40% of its maximum bandwidth, and the iPSC/2 has a peak data rate of about 2247 Kb/s or about 81% of its bandwidth. The iPSC/860 has a peak data rate of 2605 Kb/s or about 93% of its bandwidth. Also evident in the curve for the iPSC/1 is the distinct discontinuity at the 1024-byte message size. Recall from section 2 that the iPSC/1 breaks messages larger than 1024 bytes into 1024-byte segments. The iPSC/860 curve illustrates the faster message-passing for messages less than 100 bytes. The tables in Appendix A detail the data exhibited in the figures.

The dotted lines in *Fig. 2* show the data rates when messages are echoed from a node six hops away. (The tables in Appendix A give data rates for fewer hops.) The curves for the Ncube 3200 and iPSC/1 are what would be expected from a store-and-forward network, with the data rate decaying in proportion to the number of hops. The Ncube 6400, iPSC/2, and iPSC/860 hypercubes use special routing hardware, relieving the node CPU of any routing overhead and greatly reducing the penalty for multi-hop messages. For the Ncube 6400, each additional hop adds about two microseconds to the transmission time. For the iPSC/860, each hop adds about eleven microseconds for messages less than 100 bytes, and about 33 microsec-

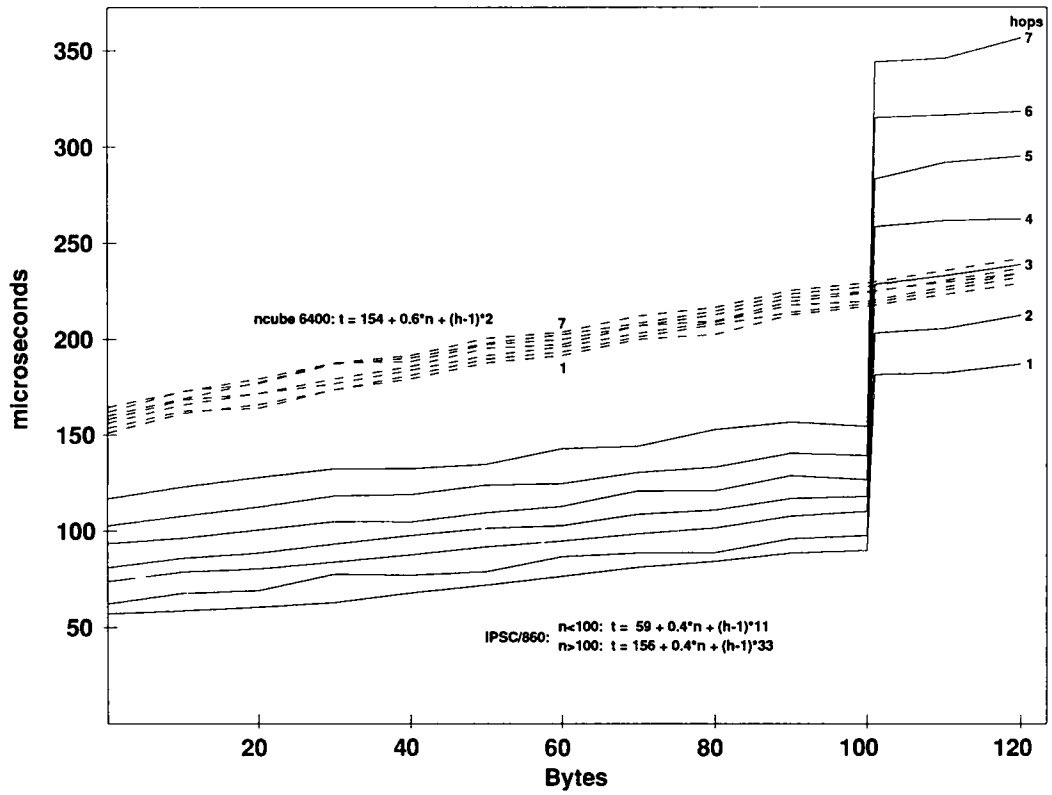

Fig. 3. Echo-test message latency.

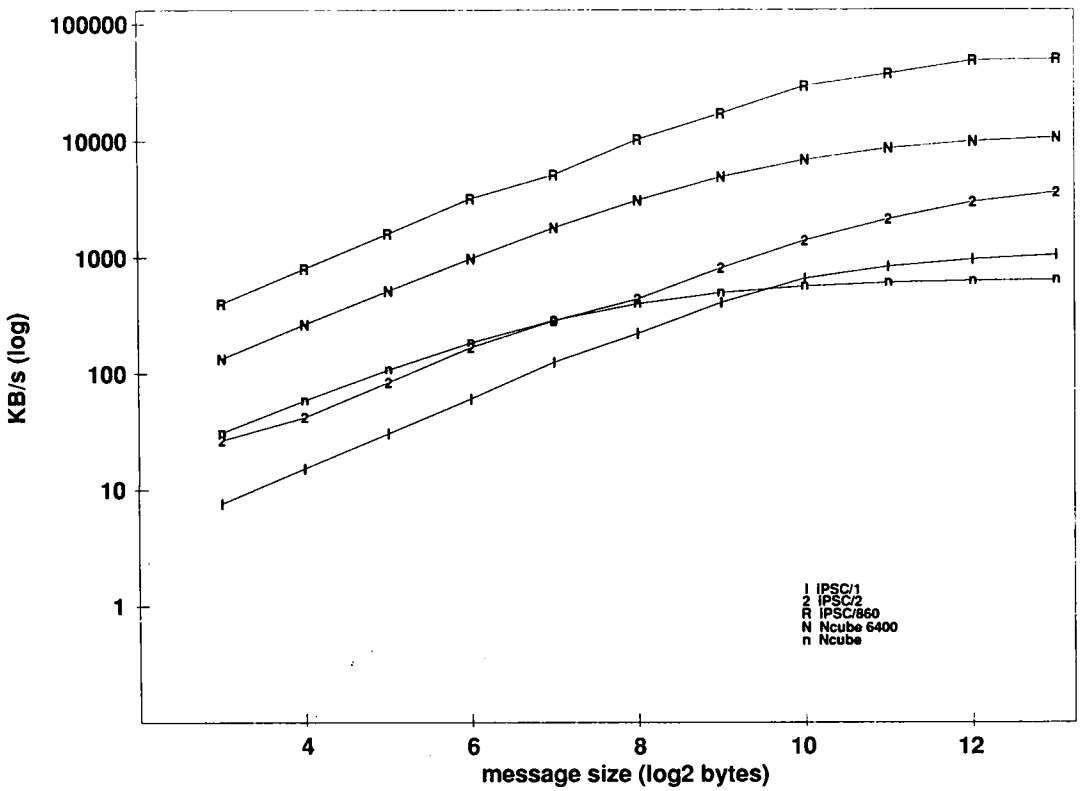

Fig. 4. Node-to-self data rates.

onds for messages greater than 100 bytes. *Figure 3* shows the average message latencies for the Ncube 6400 and iPSC/860. With the second generation routing hardware, the nodes can almost be treated as if they were directly connected.

Measuring the time it takes a node to send a message to itself can give a rough estimate of the amount of software overhead involved in message management, since no actual data transmission is required. *Figure 4* shows the data rates for a node sending a message to itself for different size messages. The overhead in passing a message from one node to another is made up of several components, some fixed and some proportional to the size of the message. Typical components are: the application gathering the data into a contiguous area, overhead in performing the call to the message-passing subroutine, context switch to supervisor mode, buffer allocation, copying the user data to the buffer area, constructing routing and error checking envelopes, obtaining the communication channel, DMA transfer with memory cycle stealing, and interrupt processing on transmission completion. The receiving node must obtain buffers for message receipt, usually initiated by an interrupt request, receive the data via DMA cycle stealing, copy the data to the user area, or, if it is a message to be forwarded and if routing is done with software, obtain a channel and initiate a DMA output request. To this is added the delay due to the actual transmission on the hardware medium, delays due to contention for the media, and delays due to synchronization and error checking acknowledgements. For segmented address spaces, like the 80286, additional overhead may be incurred for segment crossings. One or both of the DMA's may directly access the user data area, eliminating a data copy operation.

Empirically, for all five hypercubes, the communication time for a one-hop message is a linear function of the size of the message. That is, the time T to transmit to one-hop message of length N is

$$T = \alpha + \beta N$$

where α represents a fixed startup overhead and β is the incremental transmission time per byte. *Table 6* shows the startup and transmission time coefficients that were calculated from a least-squares fit of the echo data for single-hop messages. As we have seen, actual transmission times are affected by message segmentation, buffer management, and acknowledgement policy. The fixed message-passing times for small messages on the iPSC/1 system suggest that messages are being padded up to some minimum packet size of 32 or 64 bytes. The Intel iPSC/2 and iPSC/860 have a smaller startup time for messages of 100 bytes or less (e.g. a startup time of only 75 microseconds for the iPSC/860).

We also used the echo test to measure the performance of host-to-node communications. The test was performed with the corner node (node 0) for the iPSC/860, iPSC/2, and Ncube machines. Node 0 was also used for the iPSC/1, though all iPSC/1 nodes are attached to a global communication channel with the host. *Figure 5* shows host-to-node data rates for various message sizes. The host-to-node performance of the iPSC/1 is nearly six times slower than Ncube 3200 and is nearly ten times slower than the iPSC/1's node-to-node speeds for large messages. The iPSC/2 and iPSC/860 reach host-to-node speeds of nearly one million bytes per second. The relative performance of a vendor's node-to-node and host-to-node

Table 6
Least-squares estimate of communication coefficients

	Coefficients of communication microseconds				
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200
Startup (α)	136 (75)	862.2	697 (390)	200	383.6
Byte transfer (β)	0.4	1.8	0.4	0.6	2.6

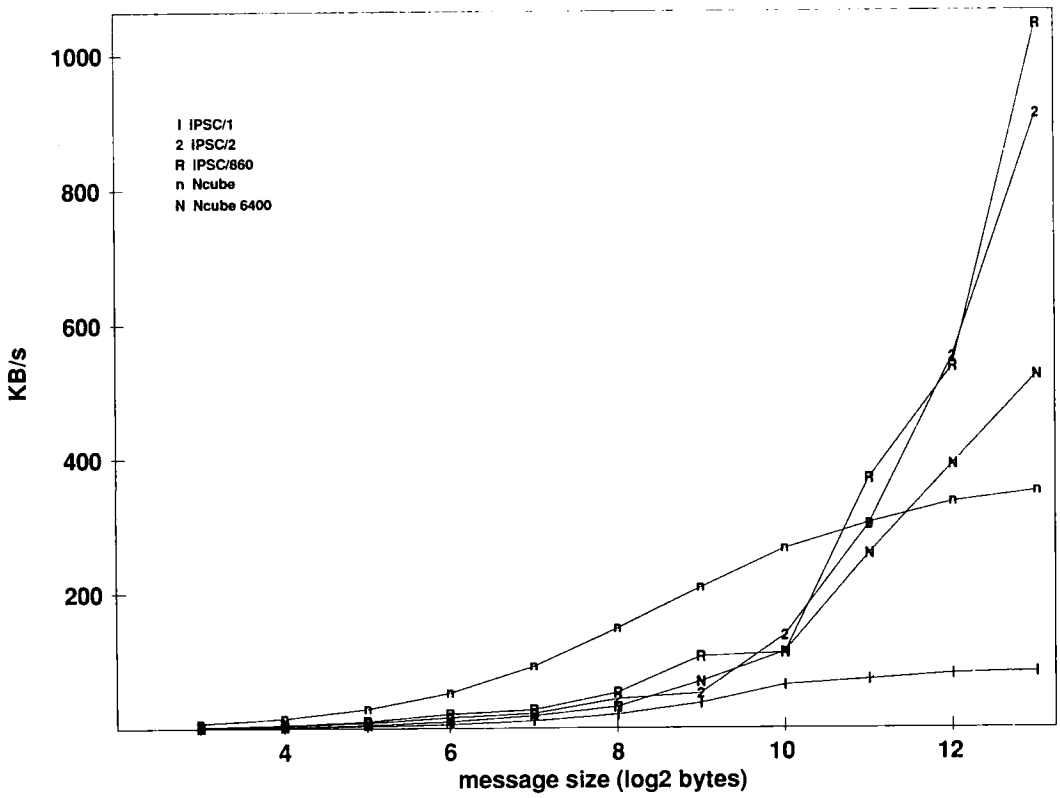

Fig. 5. Host-to-node data rates.

communications clearly should affect the extent to which the host participates in a problem solution.

4.2. Routing overhead

Two tests were constructed to measure the interaction of computation with communication on the Intel and Ncube hypercubes. In the first test, an echo test was run between two nodes that were two hops apart. The routing node between the two nodes was running an application level program that was executing an infinite loop. In fact, for both Intel and Ncube hypercubes, the routing algorithm is such that the return path of the echo message is different from the initial message path, thus two routing nodes participate. With both routing nodes running the infinite loop, data rates for the two-hop echo were calculated for various message sizes. The data rates were the same as measured when the routing nodes were idle. Thus the computing an application might be doing on a node will have no effect on the communication throughput of the node. This is due to the high priority given to communication interrupts on the first generation hypercubes. On the Ncube 6400, iPSC/2, and iPSC/860, routing is handled by a dedicated communication processor on each node.

A second test was constructed to measure the effect that routing messages had on node computing speed. First, the time for a node program to spin a loop N times was measured with no communications. The node program was then run on the routing nodes of the two-hop echo test. The execution times for the loop were measured for various message sizes of the echo test. *Figure 6* shows the degradation in computing speed due to routing for various message sizes for

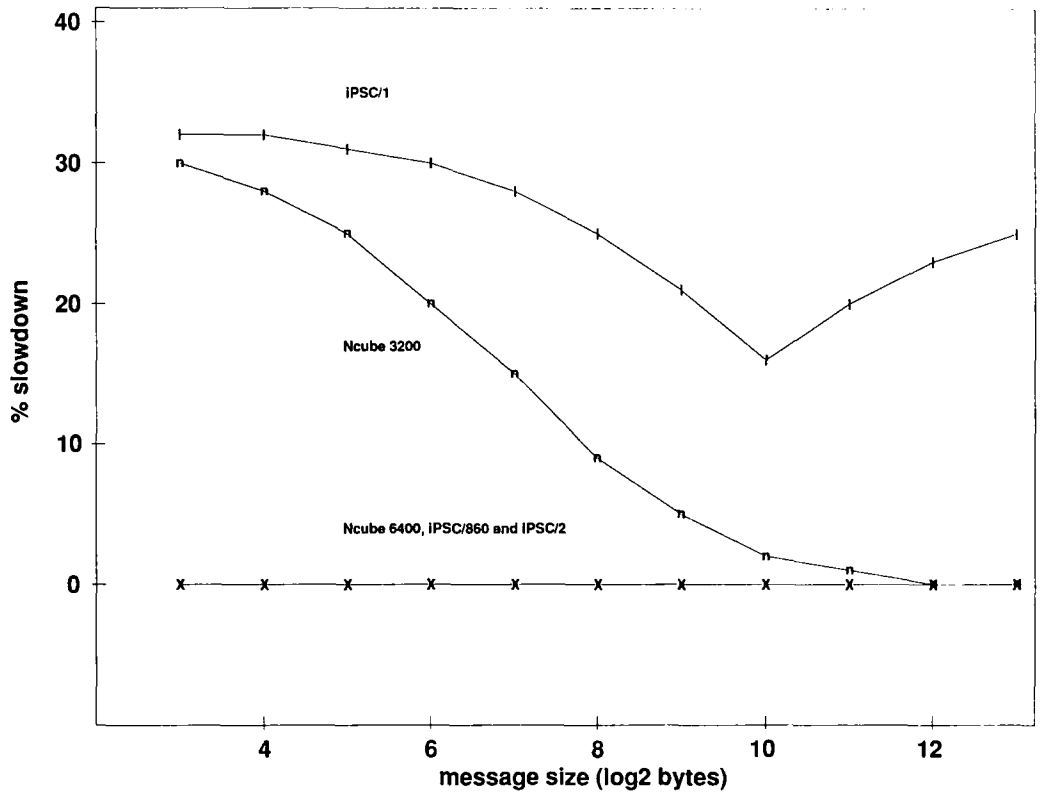

Fig. 6. Application slowdown due to routing.

both the Ncube and Intel hypercubes. The vertical axis is the percentage the loop program slowed down from its speed with no communication. For small messages, the iPSC/1 and Ncube 3200 hypercubes exhibit about a 30% loss in 'application' computation speed. As the message size increases, the interrupt rate from incoming messages decreases, and the slowdown diminishes. For the iPSC/1 hypercube, the interrupt rate increases again for messages larger than 1024 bytes, since the iPSC/1 breaks messages into 1024-byte packets. However, the Ncube 6400, iPSC/2, and iPSC/860 show no loss in computation speed, since routing is handled by independent communication hardware.

4.3. Contention

All of the communication data rates that we have reported have been measured on idle hypercubes. In actual applications, other message traffic may compete for the communication channels, either from the application itself or from applications in other subcubes. Other sub-cubes may need to use another sub-cube's communication channels to reach the host processor, I/O processor, or other service nodes. The iPSC/2, iPSC/860, and Ncube 6400 use circuit-switching to manage the communication channels. When a message is to be sent, a header packet is sent to reserve the channels required. When this 'circuit' is established, the message is transmitted, and an end-of-message indicator releases the channels.

A program was developed to measure the effect of contention on the data rate of a communication channel. The program has node 0 continuously send messages to node 7. The messages from node 0 to node 7 pass through node 1 and node 3. The amount of load presented by node 0 is varied by selecting various message sizes. With a communication load from node 0 to node 7, node 1 then sends a stream of messages to node 3. Node 3 measures the

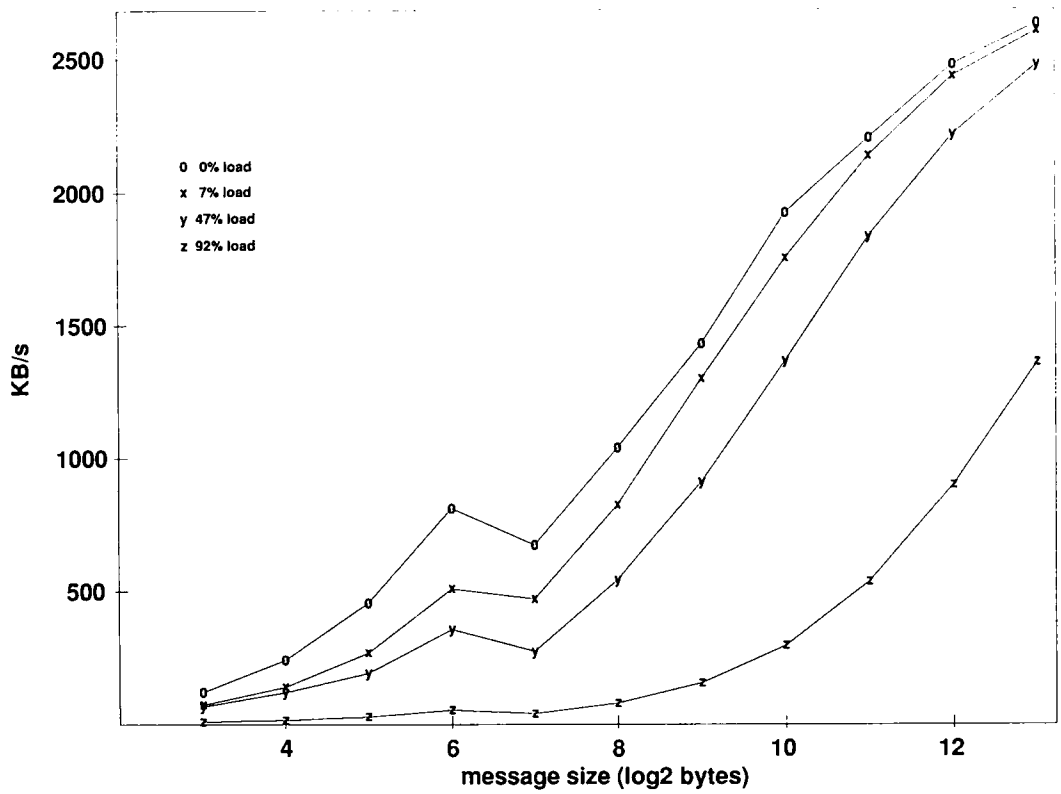

Fig. 7. Data-rates for the iPSC/860 with channel contention.

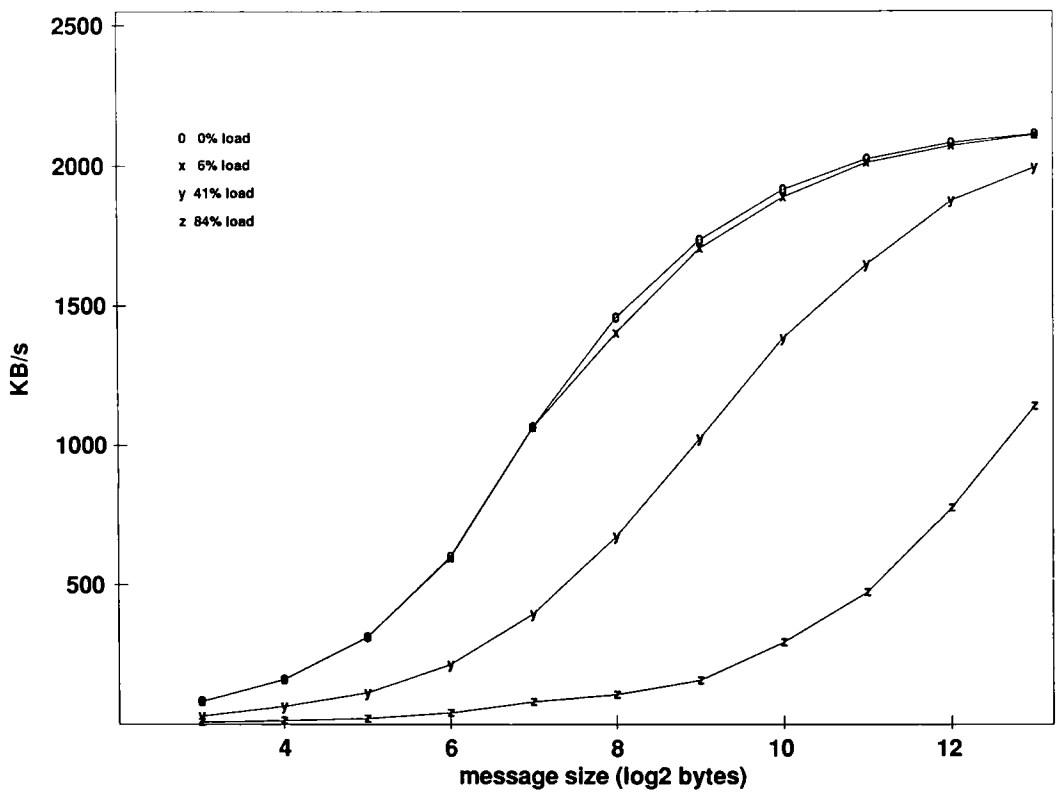

Fig. 8. Data-rates for the Ncube 6400 with channel contention.

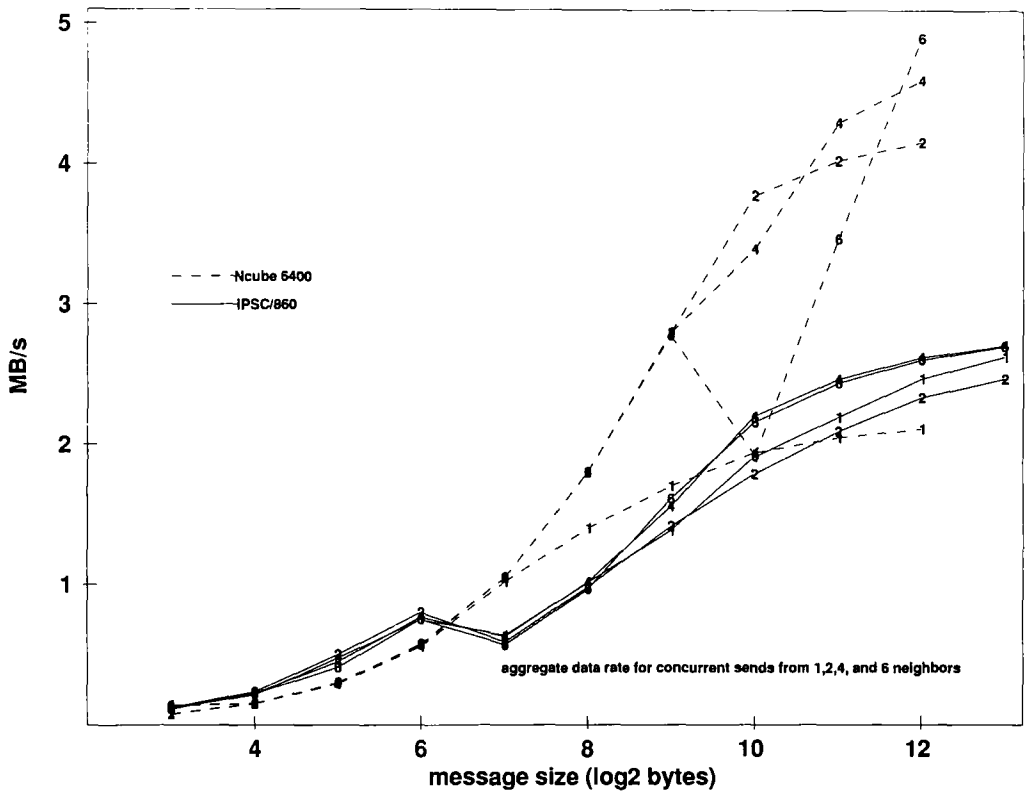

Fig. 9. Receive data rates for concurrent sends.

data rate of messages arriving from node 1 under various loads and for various message sizes. Both the iPSC/860 (*Fig. 7*) and the Ncube 6400 (*Fig. 8*) exhibit the expected behavior, as the load from node 0 to 7 increases, the data rate from node 1 to node 3 decreases. The effect of contention can vary from run to run and can slow down an application. Bokhari [1] reports more extensive contention measurements on the iPSC/860.

4.4. Concurrent communication

The message-passing performance of a node may be improved by utilizing more than one of its communication channels at the same time. A fan-in test was constructed to measure the aggregate data rate of a single node when one or more of its nearest neighbors are sending it messages. *Figure 9* shows the aggregate receive data rate for various size messages when 1, 2, 4, and 6 nearest neighbors send concurrently. The iPSC/860 shows only a slight improvement in data rate as more neighbors send messages. The data rate is still bound by the maximum single-channel data rate of 2.8 million bytes per second. Even though the iPSC/860 channels can operate concurrently, only one channel can be active with the node memory buffers at any given time [13]. In contrast, the data rate measured by the receiving Ncube 6400 node increases markedly as additional nearest neighbors transmit to it concurrently.

5. File system benchmarks

Most applications require some access to secondary storage. Some computationally intensive applications, such as mapping the human genome or global climate modeling, require fast

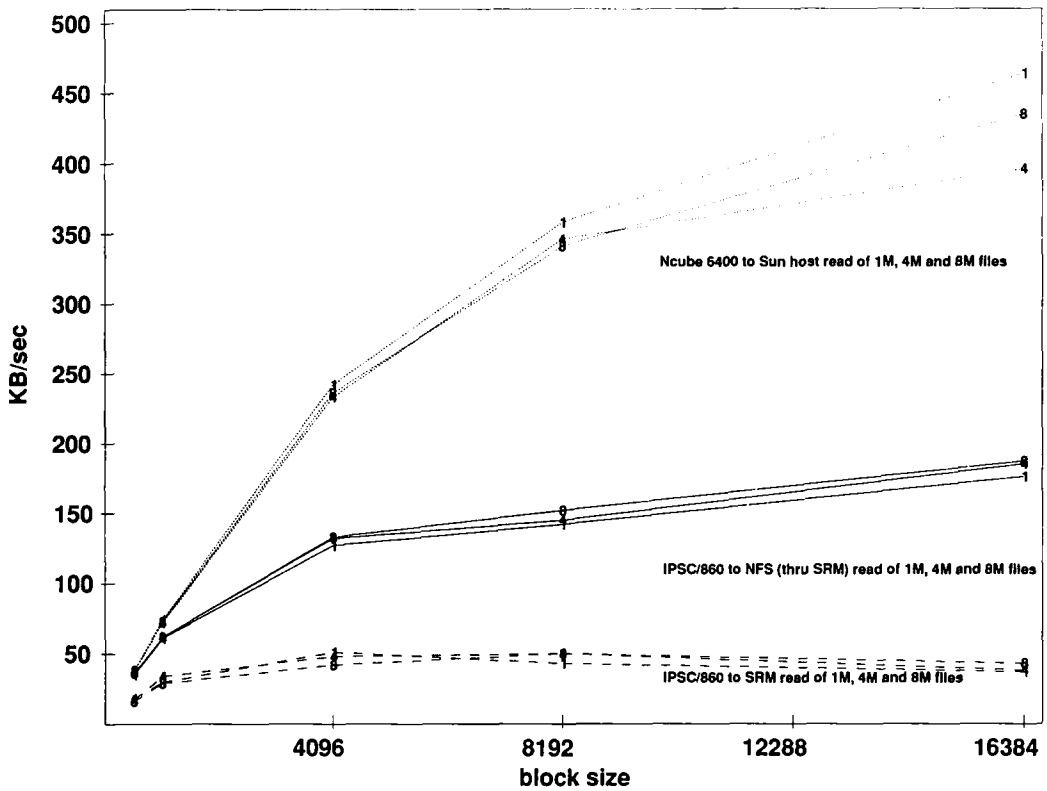

Fig. 10. Node-to-host read throughput.

access to large amounts of secondary storage. Secondary storage for hypercube systems is usually provided through the attached host processor. The run-time environment on the Intel and Ncube hypercubes permit the application programmer to use the conventional I/O constructs of C and Fortran to read and write files on the host processor. *Figure 10* shows the data rates for reading various sized host files from a hypercube node on the iPSC/860 and Ncube 6400. The iPSC/860 host used in the test was an Intel 310, an 80386-based processor with 8 million bytes of memory running System V version 3.2 and NFS. Data rates for reading from the local 340 million byte SCSI drive are much worse than reading from an NFS-mounted file system (mounted on a Sun 4/390). The Ncube 6400 host used in the test was a Sun 4/390. Performance is affected by the host-to-node communication speed, file management (buffering, blocking), and disk speed.

To avoid accessing files from or through the host processor, both the Intel iPSC/2 and iPSC/860 support a concurrent disk subsystem attached directly to the hypercube. The concurrent file system (CFS) consists of one or more I/O nodes, each with one or more disks. The I/O nodes are 80386-based processors with 4 Mb of memory and a SCSI disk interface, providing roughly 500 Mb of disk storage per drive. Each I/O node is attached to one of the hypercube compute nodes. A file created on the subsystem is striped across all of the drives using 4 Kb blocks. The node program can read and write files to the I/O subsystem in the same manner used to access files on the host processor.

Figure 11 shows the aggregate read throughput of CFS when one or more compute nodes are reading different files concurrently using one or more I/O nodes. The read throughput of CFS from a single node is an order of magnitude faster than the node-to-host file system data rates. The read throughput for a single node decreases when multiple nodes are reading from the

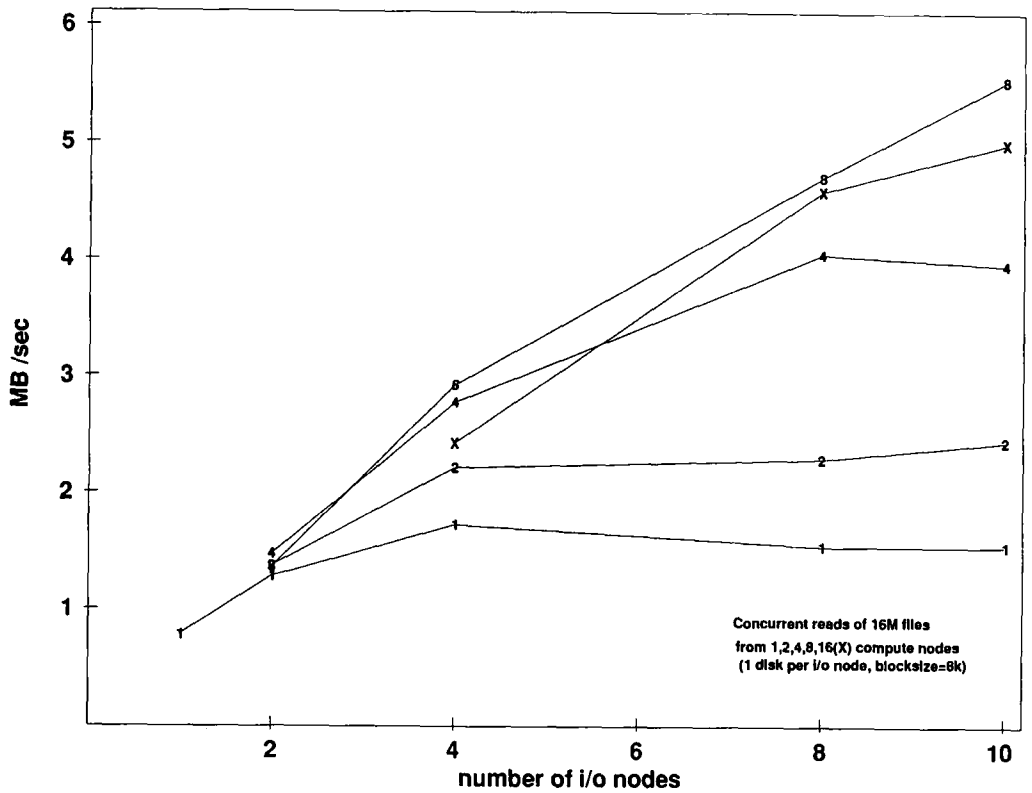

Fig. 11. CFS read throughput.

CFS. Furthermore, when the number of compute nodes doing I/O exceeds the number of I/O nodes, aggregate throughput decreases as well. Throughput improves with larger block sizes, and write times improve if the file space is pre-allocated with *lsize( )*.

6. Conclusions

The Ncube 6400 and Intel iPSC/860 show marked performance improvements over the earlier hypercubes, providing an increase in both communication and computation speeds and providing increased memory capacity and high-speed routing. *Table 7* summarizes the performance characteristics of the five hypercubes. The data rates represent the 8192-byte transfer

Table 7
Summary performance figures

	Figures of merit				
	iPSC/860	iPSC/1	iPSC/2	N6400	N3200
Data rate (KB/s)	2605	504	2247	1558	381
Megaflops	18	0.04	0.29	2.5	0.14
8-byte transfer time (μs)	80	1120	390	161	401
8-byte multiply time (μs)	0.08	43.0	6.6	1.5	13.5
Comm./Comp.	1000	26	59	107	30

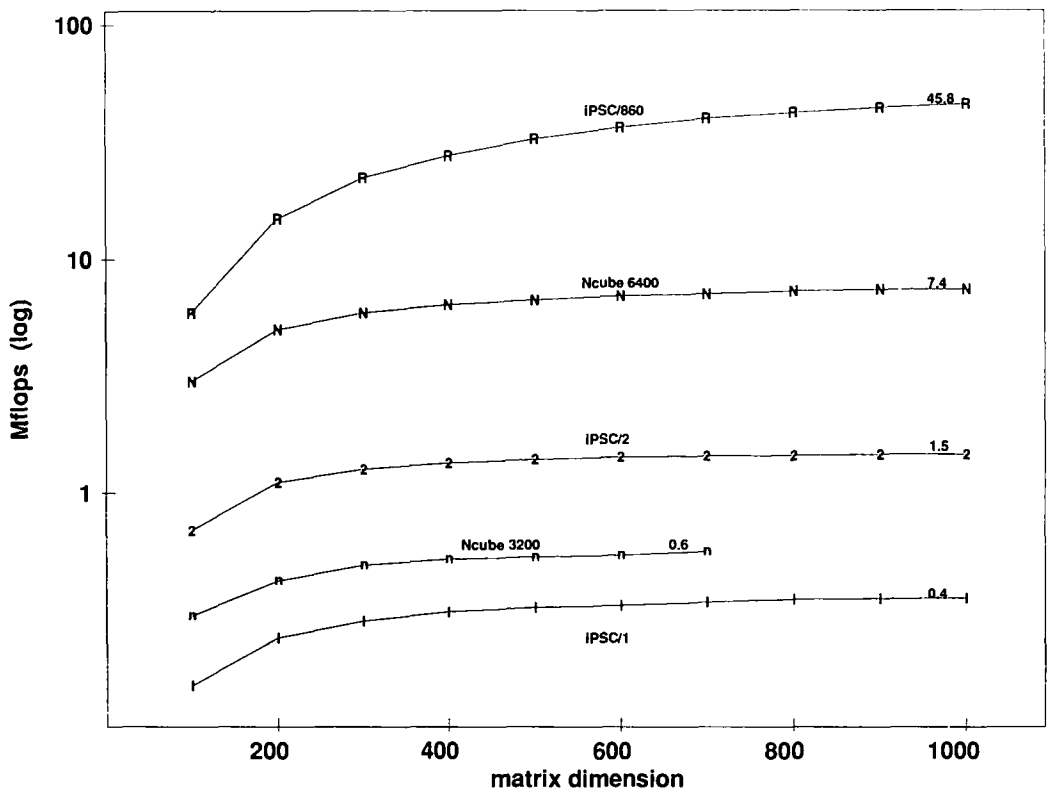

Fig. 12. Cholesky factorization megaflops for $n \times n$ matrix (16 nodes).

speeds, and the megaflops rate is calculated from compound expression results of the Caltech suite. The 8-byte transfer time is based on the 8-byte, one-hop, echo times. The structure of a hypercube algorithm will be dictated by the amount of memory available on a node, the host-to-node communication speed, and the ratio of communication speed to computation speed. As can be seen from the table, the Ncubes, iPSC/1, and iPSC/2 have roughly equivalent communication-to-computation ratios. (The ratio was calculated using the 8-byte transfer and multiply times.) The ratio on the iPSC/860 (due to its much faster CPU) suggests the need for coarser grained parallelism.

As a supplement to the component performance results presented so far, *Figure 12* illustrates the aggregate performance in megaflops of the five hypercube systems in performing a single-precision Cholesky factorization of an $n \times n$ matrix in C [7]. The Ncube 3200 system was a 7 MHz system with only 128 Kb per node. The aggregate performance for a 1000×1000 matrix using 16 nodes was 45.8 megaflops for the iPSC/860 compared with 7.4 megaflops for the Ncube 6400. As another aggregate test a parallel, double precision, Fortran implementation of SLALOM (a program to solve a radiosity problem that includes file I/O [8]) ran at 15.6 megaflops on a 64-node Ncube 6400 and at 134.8 megaflops on a 64-node iPSC/860. The performance of these applications is consistent with component timings in the preceding sections.

Appendix A Data rate tables

Intel iPSC/860 communication speeds
Kb/s

Length	Host	Self	1 hop	2 hops	3 hops	4 hops	5 hops	6 hops
8	2.4	400	100.0	94.1	84.2	76.2	69.6	64.0
16	4.6	800	200.0	188.2	160.0	152.4	133.3	123.1
32	9.8	1600	376.5	336.8	304.8	266.7	256.0	237.1
64	20.7	3200	640.0	581.8	533.3	492.3	457.1	412.9
128	26.9	5120	581.8	512.0	449.1	412.9	365.7	341.3
256	52.2	10240	914.3	853.3	764.2	701.4	640.0	602.4
512	105.6	17077	1365.3	1280.0	1177.0	1101.1	1034.3	966.0
1024	110.7	29257	1780.9	1706.7	1612.6	1539.9	1462.9	1412.4
2048	369.0	37236	2111.3	2058.3	1988.4	1941.2	1869.3	1836.8
4096	535.4	48188	2423.7	2388.3	2347.3	2301.1	2269.3	2232.2
8192	1043.6	49649	2604.8	2576.1	2556.0	2528.4	2509.0	2486.2

Ncube 6400 communication speeds
Kb/s

Length	Host	Self	1 hop	2 hops	3 hops	4 hops	5 hops	6 hops
8	0.8	134.4	49.8	49.2	48.3	47.9	46.9	46.8
16	2.1	265.9	98.0	97.3	95.4	93.9	93.3	91.9
32	4.6	515.5	183.1	181.2	179.2	177.3	174.8	172.4
64	9.3	980.4	324.7	323.6	320.5	315.5	313.5	309.6
128	18.1	1801.8	542.0	537.6	530.5	529.1	520.8	518.1
256	31.5	3100.8	809.7	804.8	796.8	792.1	792.1	782.8
512	68.5	4907.9	1084.0	1073.8	1066.7	1066.7	1049.9	1054.0
1024	112.4	6866.9	1289.3	1287.2	1274.9	1283.1	1275.9	1268.8
2048	257.5	8625.3	1426.7	1426.6	1426.5	1426.5	1420.3	1423.3
4096	389.9	9861.3	1509.4	1512.3	1510.2	1509.1	1505.5	1508.7
8192	523.7	10631.2	1558.1	1556.4	1556.2	1554.6	1554.3	1553.6

Intel iPSC/2 communication speeds
Kb/s

Length	Host	Self	1 hop	2 hops	3 hops	4 hops	5 hops	6 hops
8	1.9	26.2	20.5	20.0	19.5	18.8	18.6	18.2
16	3.9	42.1	40.5	40.0	39.0	37.6	36.8	36.3
32	7.9	84.2	80.0	79.0	78.0	74.4	73.5	71.9
64	15.8	168.4	160.0	156.1	152.4	147.1	145.4	143.8
128	21.7	232.8	172.9	166.2	166.0	153.3	148.8	143.8
256	43.4	441.4	324.1	316.0	306.5	289.3	282.8	275.3
512	66.0	812.7	581.8	568.9	547.6	527.8	514.6	499.5
1024	139.2	1402.8	961.5	939.5	922.5	886.6	867.8	849.8
2048	301.1	2133.4	1427.2	1397.9	1379.1	1338.6	1317.0	1296.2
4096	546.1	2989.8	1887.6	1870.3	1845.0	1812.4	1788.7	1773.2
8192	910.2	3608.8	2247.5	2238.3	2220.0	2193.3	2181.6	2164.3

Intel iPSC/1 communication speeds
Kb/s

Length	Host	Self	1 hop	2 hops	3 hops	4 hops	5 hops	6 hops
8	0.7	7.6	7.1	5.0	3.7	2.9	2.4	2.1
16	1.3	15.3	14.2	10.0	7.4	5.8	4.9	4.2
32	2.7	30.5	28.4	19.7	14.5	11.5	9.6	8.2
64	5.3	70.0	55.6	37.1	28.1	22.3	18.4	15.8
128	10.4	116.4	108.9	70.1	51.7	41.3	34.4	29.3
256	19.8	222.6	196.9	124.9	91.4	72.1	59.9	50.9
512	36.6	409.6	320.0	202.8	147.3	115.7	95.2	80.9
1024	63.2	660.7	481.9	292.6	211.1	165.1	135.6	114.7
2048	71.4	835.9	494.5	405.5	318.7	263.4	216.1	190.5
4096	79.8	963.8	501.0	489.1	421.1	369.0	321.9	296.8
8192	82.7	1050.1	504.1	546.1	501.4	462.8	424.4	401.1

Ncube 3200 communication speeds
Kb/s

Length	Host	Self	1 hop	2 hops	3 hops	4 hops	5 hops	6 hops
8	7.4	30.9	19.8	13.5	10.2	8.2	6.9	5.9
16	14.5	48.9	37.8	25.7	19.5	15.7	13.1	11.3
32	28.1	107.9	68.5	46.9	35.7	28.8	24.1	20.7
64	51.7	185.2	116.3	80.0	61.1	49.4	41.4	35.7
128	91.1	289.0	179.1	124.0	95.0	76.9	64.6	55.7
256	147.2	401.8	245.5	171.3	131.4	106.6	89.9	77.5
512	207.3	498.4	300.2	211.3	162.5	132.3	111.5	96.3
1024	265.5	565.8	338.2	238.8	184.5	150.3	126.8	109.7
2048	303.3	607.1	361.5	255.7	197.8	161.3	136.2	117.9
4096	334.5	630.5	373.8	264.9	205.2	167.5	141.4	122.4
8192	349.7	642.6	380.6	269.6	209.2	170.7	144.2	124.8

References

[1] S. Bokhari, Communication overhead on the Intel iPSC-860 Hypercube, ICASE Interim Report 10, NASA Langley, 1990.

[2] H.J. Curnow and B.A. Wichman, A synthetic benchmark, *Comput. J.* 19 (1976) 87–93.

[3] J. Dongarra, Performance of various computers using standard linear equations software in a Fortran environment, University of Tennessee, CS-89-85, January 1991.

[4] T.H. Dunigan, Performance of the Intel iPSC/860 Hypercube, Tech. Rept. ORNL/TM-11491, Oak Ridge National Laboratory, Oak Ridge, TN, 1990.

[5] T.H. Dunigan, A message-passing multiprocessor simulator, Tech. Rept. ORNL/TM-9966, Oak Ridge National Laboratory, Oak Ridge, TN, 1986.

[6] T.H. Dunigan, Hypercube clock synchronization, Tech. Rept. ORNL/TM-11744, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.

[7] G.A. Geist and M.T. Heath, Matrix factorization on a hypercube multiprocessor, in: *Hypercube Multiprocessors 1986*, M.T. Heath, ed. (SIAM, Philadelphia, 1986) 161–180.

[8] J. Gustafson, personal correspondence, Ames Lab, 1991.

[9] A. Kolawa and S. Otto, Performance of the Mark II and Intel Hypercubes, in: *Hypercube Multiprocessors 1986*, M.T. Heath, ed. (SIAM, Philadelphia, 1986) 272–275.

[10] Intel, iPSC User's guide, Intel 17455-03, Portland, Oregon, October, 1985.

[11] Ncube, Ncube 6400 processor manual, Ncube V1.0, Beaverton, OR, 1990.

[12] Ncube, Ncube handbook, Ncube, V1.1, Beaverton, OR, 1986.

[13] S.F. Nugent, The iPSC/2 direct-connect communication technology, in: *Proc. Third Hypercube Conf. CACM* (1988) 51–60.

[14] C.L. Seitz, The cosmic cube, *Commun. ACM* 28 (1985) 22–33.

[15] R. Weicker, Dhrystone: a synthetic systems programming benchmark, *Commun. ACM* 27 (1984) 1013–1030.