

ornl

**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0262472 9

ORNL/TM-10400

Performance of Three Hypercubes

T. H. Dunigan

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

CIRCULATION SECTION

FORM ROOM 275

LIBRARY LOAN COPY

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to use this
report, send in name with report and
the Library will arrange a loan.

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A03 Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Engineering Physics and Mathematics Division

Mathematical Sciences Section

PERFORMANCE OF THREE HYPERCUBES

T. H. Dunigan

Date Published - May 1987

The work was supported by the
Applied Mathematical Sciences subprogram
of the Office of Energy Research,
U. S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
operated by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400



3 4456 0262472 9

Table of Contents

Abstract	1
1. Overview	1
1.1 Introduction	1
1.2 Test environment	2
2. Configurations	2
2.1 Ametek System 14	3
2.2 Intel iPSC	3
2.3 Ncube	3
3. Computation Benchmarks	4
3.1 Arithmetic tests	4
3.2 Synthetic tests	4
3.3 Memory utilization	6
4. Communication Benchmarks	6
4.1 Ring test	6
4.2 Echo tests	6
5. Routing overhead	12
6. Conclusions	14
Acknowledgements	15
References	16

Performance of Three Hypercubes

T. H. Dunigan

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

ABSTRACT

The performance of three commercially available hypercube parallel processors is analyzed. Computation and communication performance for a number of low-level benchmarks are presented for the Ametek S14 hypercube, the Intel iPSC hypercube, and the Ncube hypercube.

1. Overview.

1.1. Introduction.

This report summarizes the results of a set of benchmarks run on three commercially available hypercubes, updating earlier results in [3]. The three cubes are the first generation of the hypercube family of parallel computers, descendants of the pioneering work done at Caltech [10]. The hypercube is an ensemble of small computers interconnected in 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. The architecture is worthy of study because of the wide range of applications that are suitable for the hypercube architecture [5] and because of the attractive cost-performance of the current generation of machines.

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, and I/O (message passing). Second, a large proportion of our development work is done on hypercube simulators, both to debug and to analyze our algorithms [4]. Performance results from real hypercubes enable us to construct more effective simulators. Finally, a set of benchmarks and performance results can help us evaluate new implementations or architectures.

In the remainder of this section, we summarize the hypercube configurations and programs used in our test suite. Section 2 discusses the three hypercube architectures in more detail, emphasizing the distinctive features of each implementation. The computational power and memory capacity of the three hypercubes, and their single-channel message-passing capacity are compared in Sections 3 and 4, respectively. Section 5 illustrates the effect of communication overhead on computations. Section 6 summarizes and looks toward future work.

1.2. Test environment.

Three commercially available 64-node hypercubes were used for our benchmark suite. We have both Intel and Ncube hypercubes at Oak Ridge National Laboratory. In addition, Ametek Corporation provided us dial-in access to one of their hypercubes. The configurations utilized in the tests are summarized in Table 1.

Configurations for Tests			
	Ametek	Intel	Ncube
Number of nodes	64	64	64
Node CPU	80286/287	80286/287	custom 32-bit
Clock rate	8 MHz	8 Mhz	8 MHz
Node memory	1024K	512K	512K
Nominal data rate	3 Mbps	10 Mbps	8 Mbps
Node OS	XOS vD1	v3.0	Axis v2.3
C compiler	Lattice C v3.1	Xenix 3.4	CF&G v1.0

Table 1. Hypercube configurations used in tests.

The test programs were written in C and were run in the first quarter of 1987. The large model memory option was used with the C compiler for Intel (*-Alfu*) and Ametek (*-ml*), and stack checking was disabled for the Intel and Ncube C compilers. The test suite was selected for simplicity of 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.

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

Table 2. Benchmark programs used in tests.

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. Besides the application program, each node contains a small operating system that manages message passing.

2.1. Ametek System 14.

The Ametek System 14 consists of from 16 to 256 nodes attached to a VAX host via a 16-bit parallel interface. Only one corner of the cube, node 0, is attached to the host. The host runs DEC's ULTRIX operating system, and thus provides the full set of software management tools associated with UNIX. The hypercube nodes are Intel 80286/80287 chips running at 8 MHz with one million bytes of memory per node. The node-to-node communication channels are controlled by a separate communication coprocessor, a 10 MHz Intel 80186. Each node-to-node data channel is rated by the vendor at 3 million bits per second. Such a vendor rating implies communication shall never exceed that data rate. Only C is supported at present; Fortran eventually will be supported. Since the architecture of the host and nodes are different, a cross-compiler is provided on the VAX for developing node programs. To take advantage of the 80287, however, one must compile C programs on an IBM PC attached via a serial line. A set of command procedures are provided to make this acceptably invisible. The hypercube is presently a single-user subsystem.

The node operating system, XOS, is structured much like the Crystalline system [10]. Communication is synchronous at the application level and only between nearest neighbors. There is no implicit message routing. Message passing is based on 8-byte packets, though multipacket subroutines are provided as well. Various routines are provided for ring and mesh communications as well as full hypercube topology routines [1]. A growing set of application libraries is available. A simulator is provided on the host to assist in program debugging and analysis. Command procedures enable one to switch from simulator mode to hypercube mode with little effort.

2.2. Intel iPSC.

The Intel iPSC 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 512K bytes of memory and is attached to the host via a global communication channel. (Intel now offers a 4.5 Megabyte/node option as well as an optional vector processing node with 1 Megabyte of memory.) 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 hypercube is presently a single-user subsystem.

The node operating system supports message routing, asynchronous communications, and multi-tasking within each node [7]. 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. A growing set of application libraries is available.

2.3. Ncube.

The Ncube 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 is presently running at 8 Mhz. It is expected the chip will soon run at 10 Mhz. The chip contains both floating point and message handling facilities. It is surrounded by 512K of memory. The chip is also used as the interface processor between the hypercube and the host. The hypercube may be divided into logical subcubes for multi-user use [9].

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 application library and program development tools continue to grow. A four-node board is available for use on an IBM PC/AT.

3. Computation Benchmarks.

3.1. Arithmetic tests.

To compare our test results with earlier hypercube benchmarks performed at Caltech [8], we implemented a series of tests to measure the arithmetic speeds of the CPU for integer and 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.

Arithmetic Times				
microseconds				
	Ametek	Intel	Ncube	VAX
INTEGER*2 +	2.5	2.5	4.5	3.3
INTEGER*4 +	5.2	5.0	4.9	1.8
INTEGER*2 *	3.9	4.0	6.0	5.1
INTEGER*4 *	194.0	36.5	6.3	2.4
REAL*4 +	51.2	38.0	16.6	7.1
REAL*8 +	32.4	41.5	11.5	4.6
REAL*4 *	52.4	39.5	18.5	9.3
REAL*8 *	33.9	43.0	13.5	6.5
REAL*4 *+*+*	39.8	23.1	10.6	5.6
REAL*8 *+*+*	28.3	24.1	7.8	4.4

Table 3. Arithmetic operation times (microseconds).

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 only anomaly is the large INTEGER*4 multiplication time for the Ametek, because it uses a subroutine to perform the computation. The last two entries 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 word size of the machine. The default integer word size is 16 bits for the Intel and Ametek machines and is 32 bits for the VAX and Ncube. The degree to which the compilers comply to the C requirement varies. The Ncube is roughly three times faster than the Ametek and Intel hypercubes, operating at 0.12 megaflops to the 80287's 0.04 megaflops.

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 256,000 repetitions of the double precision floating point arithmetic operations are illustrated in Figure 1 and Table 4. In the *sieve* variables of type *register int* are used, which means 16-bit arithmetic for the Ametek and Intel machines and 32-bit arithmetic for the Ncube and VAX.

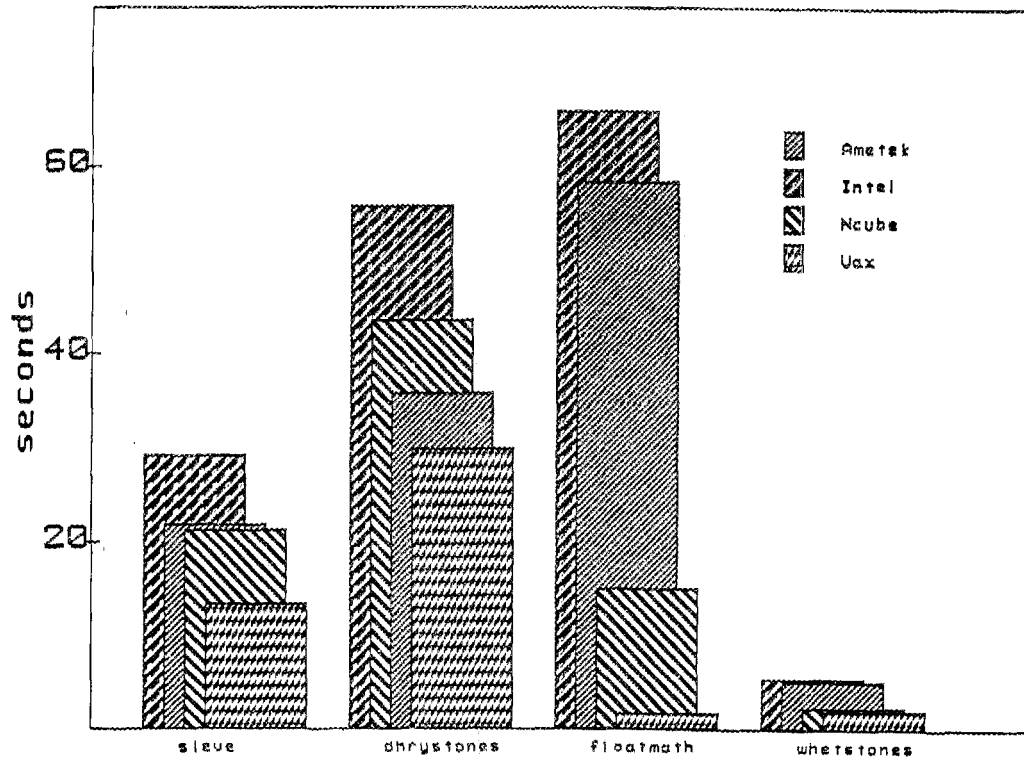


Figure 1. Synthetic computation tests.

Simple Computation Tests				
	seconds			
	Ametek	Intel	Ncube	VAX
Sieve	22.0	29.3	21.4	13.6
Floatmath	58.5	66.3	15.2	1.9
Dhrystone	35.9	55.9	43.7	30.0
Whetstone	5.3	5.6	2.5	2.2

Table 4. Execution time in seconds for various test suites.

Figure 1 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 [11]. 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 Intel and Ametek C compilers and 32-bit arithmetic for the Ncube and VAX. The figure 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 Intel C generates an additional move instruction for references to external variables which explains the slower performance of the Intel compared to the Ametek.

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 1,000 byte 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.

Memory Capacity			
K Bytes			
	Ametek	Intel	Ncube
Total	1024	512	512
Available	912	366	453

Table 5. Node memory capacity and usage.

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 (in the case of Intel and Ncube), and C run-time environment. For the 80286 architectures, memory is managed in 64K segments, so there may be additional small chunks of free memory available. The Intel user also can specify the amount of memory to use for message passing buffers; twenty buffers were specified for the memory test. As was mentioned in section 2, Intel now has a 4.5 million byte memory option, and Ncube recently expanded the memory from 128K to 512K bytes.

For any computer system, the amount of main memory is a critical metric, and there never seems to be enough. For the 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. Ring test.

As a first test of node-to-node communication speed, the time to pass a message 100 times around a 64-node Gray-code ring was measured. The Gray-code mapping ensures that a distance of only one hop is required between each node and its successor in the ring. The Ametek implementation used the *pass* message-passing primitive. *Sendw/recvw* were used on the Intel, and *nwrite/nread* were used on the Ncube. Figure 2 shows the times for messages of size 8 bytes to 8192 bytes. The Ncube is the fastest for small messages, but falls behind the Intel machine for messages larger than 256 bytes — but by less than a factor of two in both cases. The Ametek is slightly faster than the Intel for messages smaller than 32 bytes but is three times slower for messages larger than 1024 bytes. For smaller messages, times increase by only a few percent as the message size is doubled. Table 6 lists the times as well as the node-to-node data rate in bytes-per-second.

4.2. Echo tests.

To further 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 node. The test node measures the time to send and receive the message *N* times. The nodes utilized the same message-passing functions as in the ring test. Figure 3 shows the data rates for the three hypercubes over various message sizes, where the echoing node is one hop away. Consistent with the ring results, the Ncube is fastest for small messages; the Intel is fastest for large messages. The Ametek peaks out just over 100 KB/s

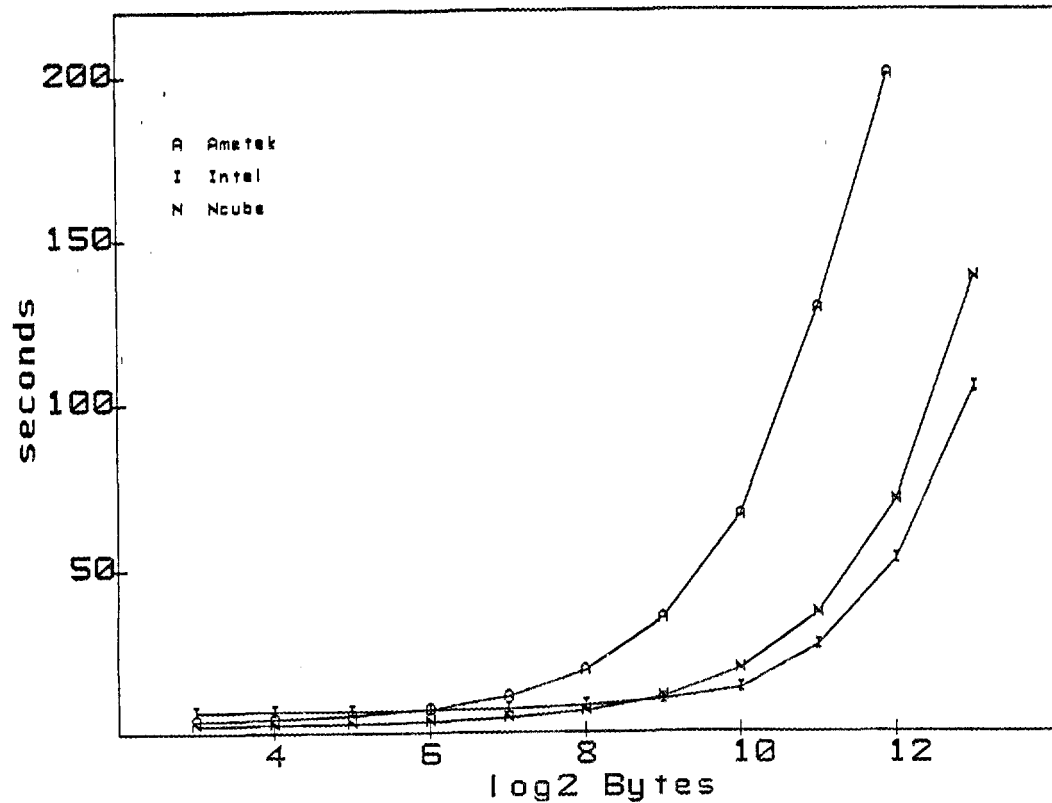


Figure 2. Times for 100 revolutions of 64-node ring.

Ring Times			
seconds (KB/s), 64-node ring, 100 revolutions			
Length	Ametek	Intel	Ncube
8	4.0(13)	6.7(8)	2.6(20)
16	4.5(23)	6.8(15)	2.7(38)
32	5.5(37)	6.9(30)	3.0(69)
64	7.4(55)	7.1(58)	3.5(117)
128	11.3(72)	7.5(109)	4.6(179)
256	19.1(86)	8.4(195)	6.7(245)
512	34.7(94)	10.1(323)	10.9(300)
1024	65.9(100)	13.6(482)	19.4(339)
2048	128.2(102)	26.5(494)	36.3(361)
4096	252.8(104)	52.3(501)	70.1(374)
8192	502.2(105)	104.1(504)	137.8(381)

Table 6. Gray-code ring times in seconds.

or about 28% of its maximum single-channel bandwidth. The Ncube levels off around 380 KB/s or about 38% of its bandwidth. The Intel levels off at around 505 KB/s or about 40% of its maximum bandwidth. The figure also shows the cross-over points where one machine performs better than another. Also evident in the Intel curve is the distinct

discontinuity at the 1024-byte message size. Recall from section 2 that Intel breaks messages larger than 1024 bytes into 1024-byte segments. Tables 8, 9 and 10 at the end of this section detail the data exhibited in the figures.

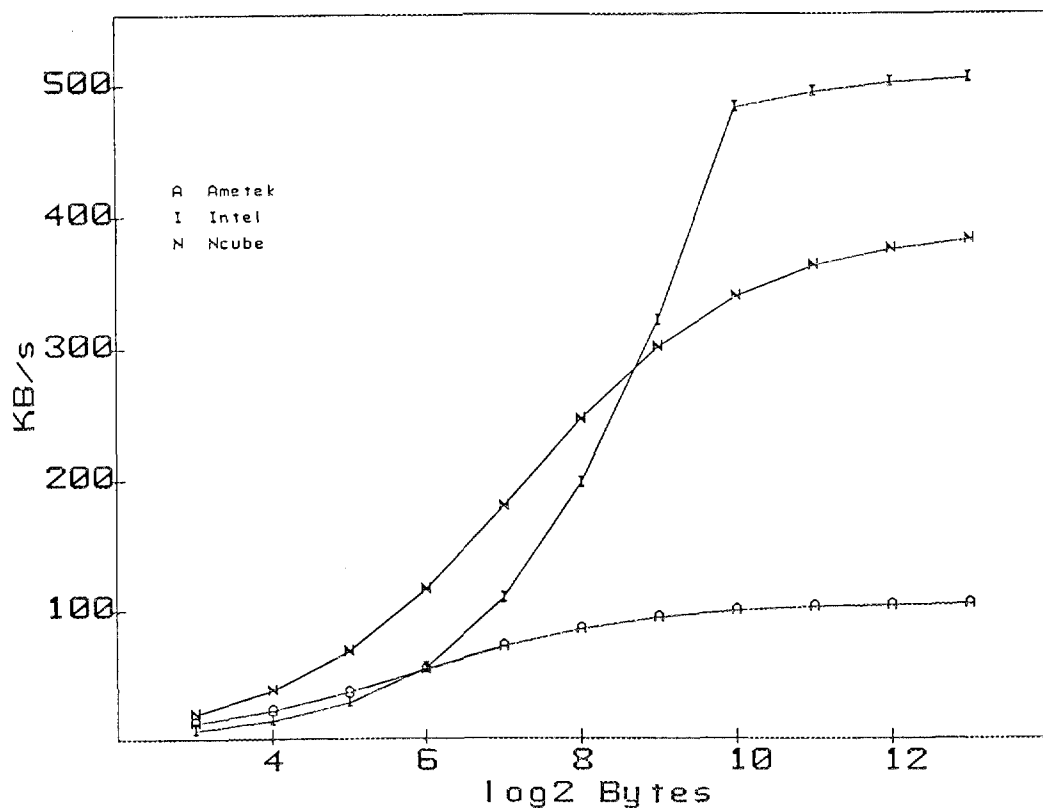


Figure 3. One-hop data rates.

The Intel and Ncube node operating systems support message routing so we can use the echo test to measure data rates for passing messages to non-adjacent nodes. Figure 4 illustrates the performance of the Intel and Ncube machines for passing messages of four different sizes to nodes from one to six hops distant. (Tables 9 and 10 give data rates for additional message sizes.) As before, Ncube is faster for smaller messages. The curves are what would be expected from a store-and-forward network, with the data rate decaying in proportion to the number of hops. Of note, however, is the Intel curve for messages larger than 1024 bytes. Segmenting the message into 1024-byte packets will yield higher data rates for multi-hop messages, since a packet may be forwarded while the next packet arrives. But due to the way Intel acknowledges forwarded versus single-hop messages, the data rate is actually higher for a message traveling two hops than for a message traveling just one hop!

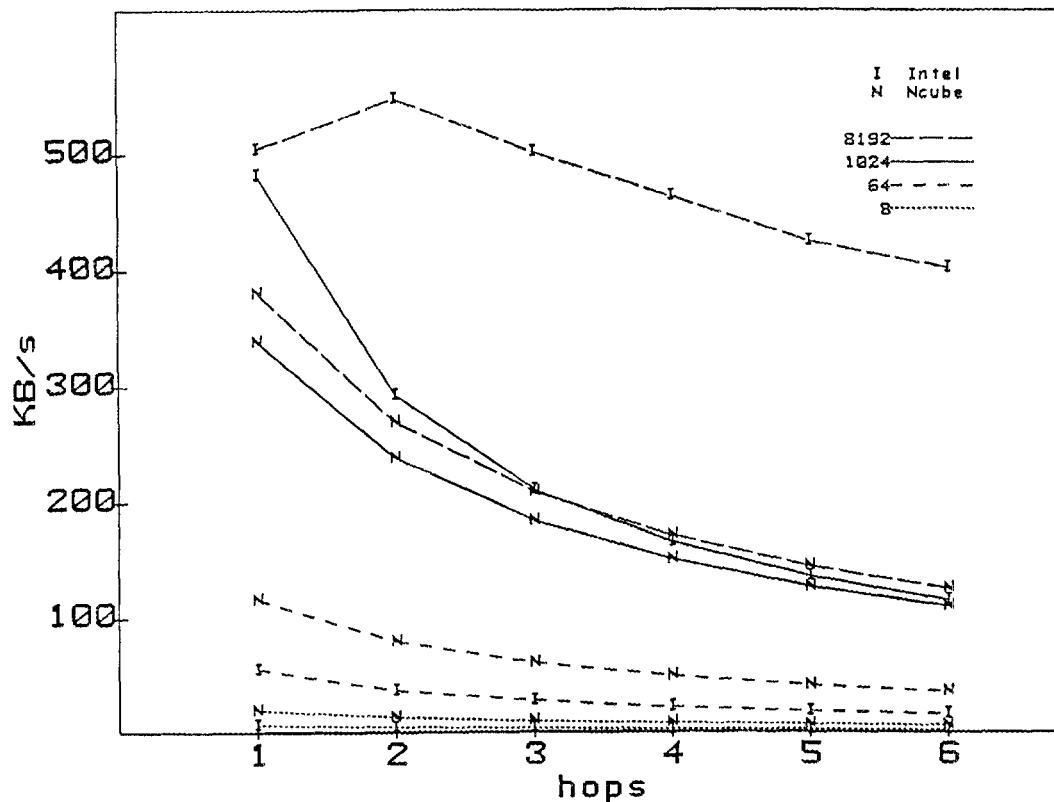


Figure 4. Multi-hop data rates.

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 5 shows the data rates for a node sending a message to itself for different size messages. The overhead in passing a message is made up of several components, some fixed and some proportional to the size of the message. Typical components are:

- the application must gather 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,
- 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, 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.

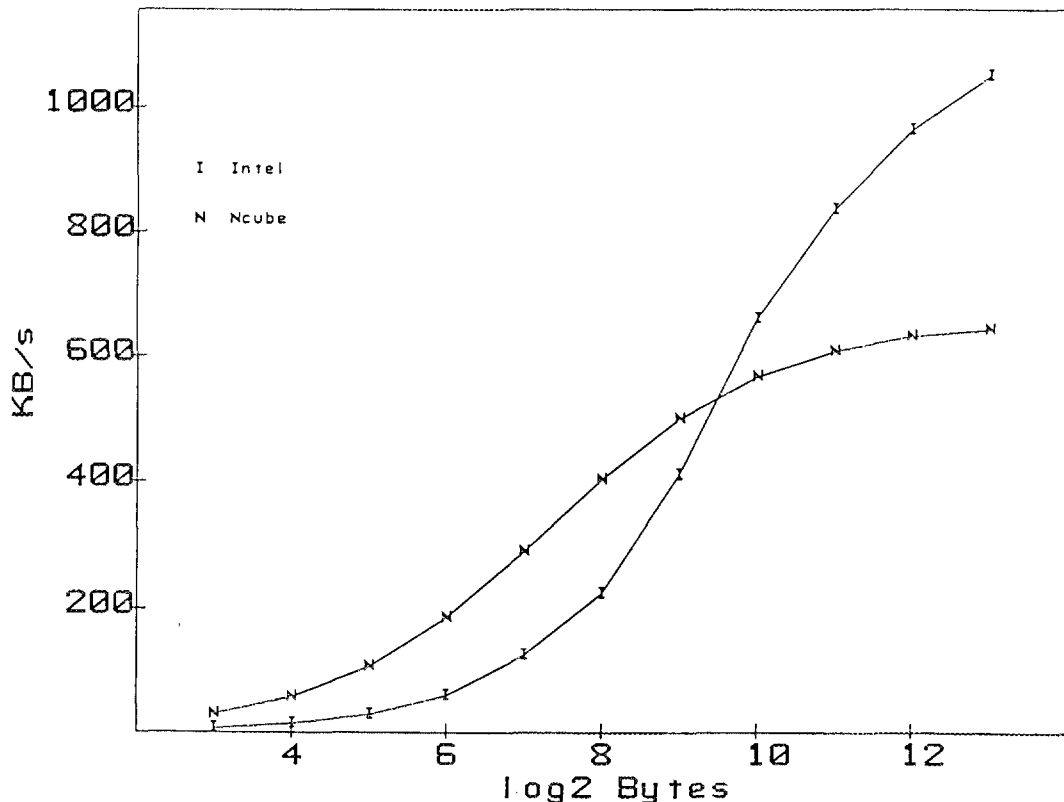


Figure 5. Node-to-self data rates.

Empirically for all three 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 a one-hop message of length N is

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

where α represents a fixed startup overhead and β is the transmission time for one byte. Table 7 shows the startup and transmission time coefficients that were calculated from a least-squares fit of the echo data for single-hop messages. The coefficients are in close agreement with data reported by by Grunwald and Reed [6] and show the improvements made in the Intel message-passing software over earlier results reported by Kolawa and Otto [8]. As we have shown, actual transmission times are affected by message segmentation, buffer management, and acknowledgement policy. The fixed message-passing times for small messages on the Intel system suggest that messages are being padded up to some minimum packet size of 32 or 64 bytes.

Coefficients of Communication			
microseconds			
	Ametek	Intel	Ncube
Startup (α)	563.5	862.2	383.6
Byte transfer (β)	9.5	1.8	2.6

Table 7. Least-squares estimate of communication coefficients.

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 Ametek and Ncube machines. Node 0 was used for the Intel test, though all Intel nodes are attached to a global communication channel with the host. The Ametek host program utilized *rdnIH* and *wtnIH* and the node program used *pass*. Figure 6 shows data rates for various message sizes. Since the Ncube uses a node CPU as its host interface to the hypercube, it is not surprising that data rates are comparable to its node-to-node performance. The Ametek 16-bit parallel interface is somewhat slower than Ncube, but is a little faster than the Ametek node-to-node speeds. Intel is nearly six times slower than Ncube and is nearly ten times slower than Intel's node-to-node speeds for large messages. One can also see the effect of the 1024-byte segments on the Intel curve. The relative performance of a vendor's node-to-node and host-to-node communications clearly should affect the extent to which the host participates in a problem solution.

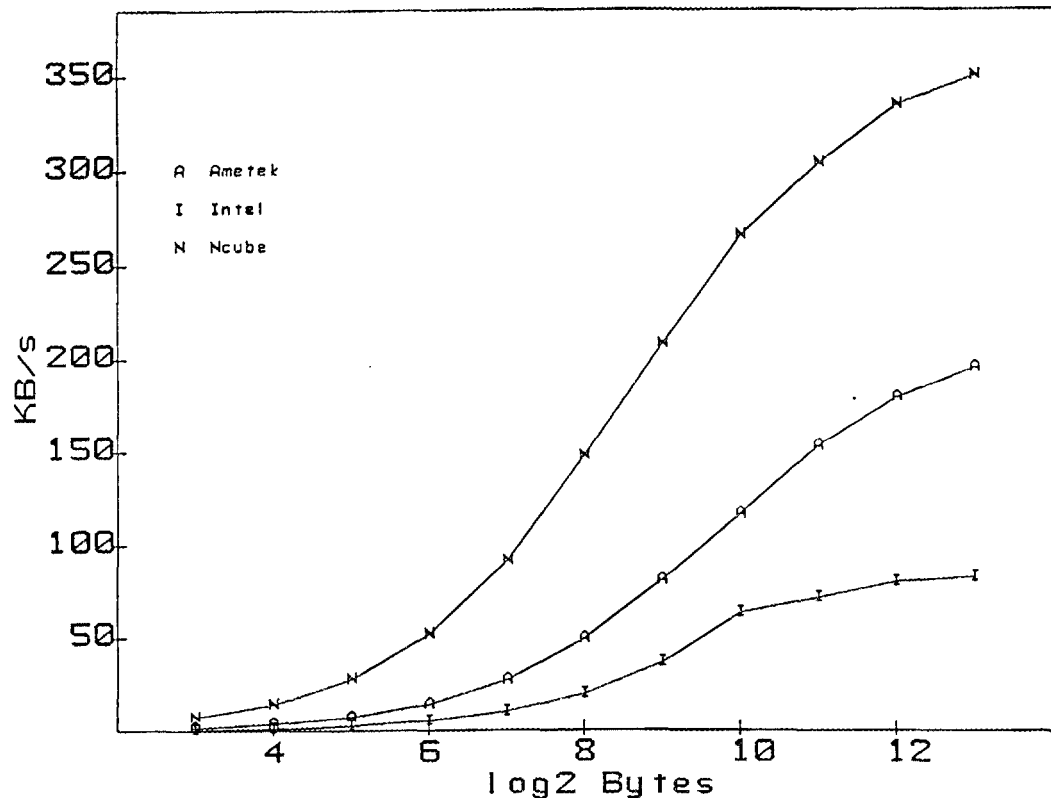


Figure 6. Host-to-node data rates.

Ametek Communication Speeds		
KB/s		
Length	Host	1 hop
8	2.0	12.5
16	3.8	22.4
32	7.5	36.9
64	14.6	54.6
128	27.4	71.8
256	49.1	85.3
512	80.1	94.1
1024	114.4	99.2
2048	152.2	102.0
4096	179.5	103.4
8192	196.9	104.2

Table 8. Ametek host-to-node and node-to-node data rates.

Intel 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

Table 9. Intel single channel data rates.

5. 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 the Intel and Ncube, 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.

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

Ncube 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	58.9	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

Table 10. Ncube single channel data rates.

message sizes of the echo test. Figure 7 shows the degradation in computing speed due to routing for various message sizes for both the Intel and Ncube hypercubes. The vertical axis is the percentage the loop program slowed down from its speed with no communication. For small messages, Intel and Ncube 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.

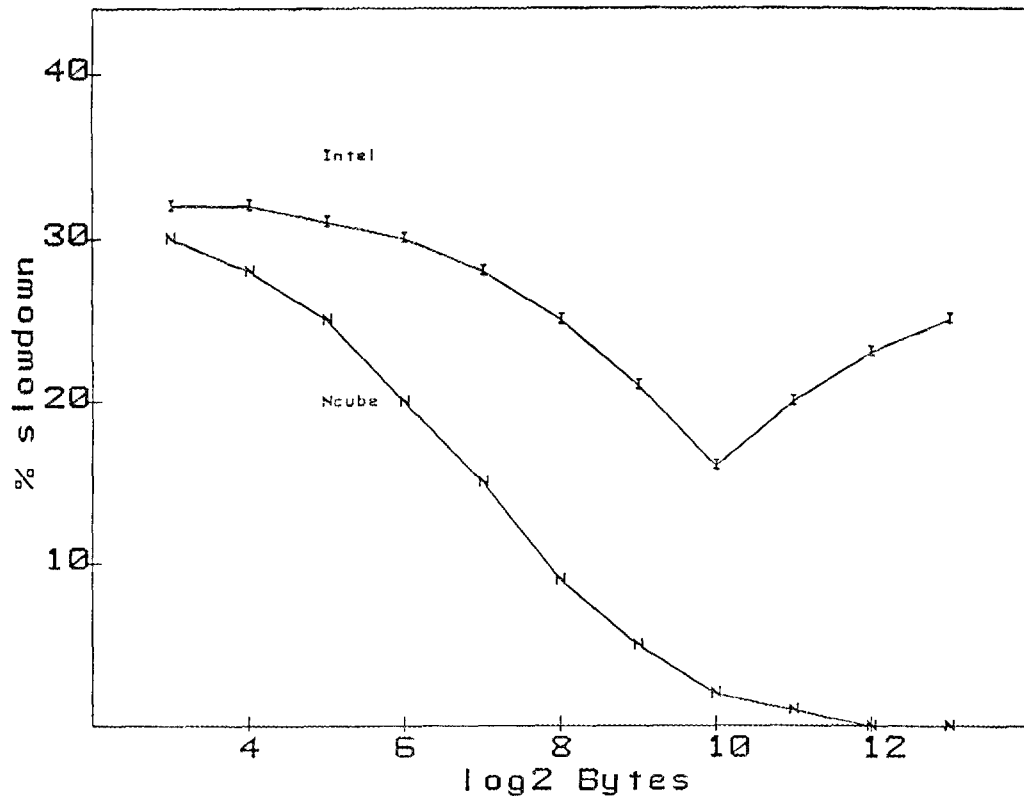


Figure 7. Application slowdown due to routing.

For the Intel hypercube, however, the interrupt rate increases again for messages larger than 1024 bytes, since Intel breaks messages in to 1024-byte packets. We have already shown that Ncube can transmit about twice as many 8-byte messages per second as Intel, thus the overhead for routing is even less for Ncube if we were to plot slowdown versus messages-per-second.

6. Conclusions.

We have shown that despite differences in hardware and software the three hypercubes have very similar performance characteristics. On the other hand, even with identical computing hardware, computation speeds will differ due to compiler and operating system differences. Table 11 summarizes the performance characteristics of the three hypercubes.

Figures of Merit			
	Ametek	Intel	Ncube
Data rate (KB/s)	104	504	381
Kiloflops	40	40	140
8-byte transfer time (μ s)	640	1120	401
8-byte multiply time (μ s)	33.9	43.0	13.5
Comm./Comp.	19	26	30

Table 11. Summary performance figures.

The data rates represent the 8192-byte transfer speeds, and the kiloflops 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 three hypercubes have roughly equivalent communication-to-computation ratios. The ratio was calculated conservatively using the 8-byte transfer and multiply times.

We believe the addition of a communication coprocessor would greatly enhance hypercube performance. Furthermore, the segmented address space of the 80286 proves somewhat of a nuisance. A 32-bit node processor would be desirable. The need for several levels of node-to-node communication support seems evident. It would be desirable to have a very high speed, small, non-routing, non-buffering communication system as well as a more robust, routing, buffering option. The robust communication scheme could be used for debugging and for applications requiring full use of the hypercube inter-connection topology. The high speed option could be used for time-critical applications.

In the future, we hope to expand the test suite in order to test the message-passing systems under heavy load and to measure the total message-handling capacity of a node using multiple channels simultaneously. Running the Caltech Crystalline operating system on the Intel hardware could provide useful performance information on the price paid for buffered, asynchronous communication with automatic routing. Since one of the main thrusts of our research is the development of algorithms for matrix computations, we anticipate running many of these algorithms on the available hypercube architectures. Performance results from these algorithms will help to measure the aggregate throughput of hypercube systems as an ensemble of processors and will supplement the component performance results of this report.

Acknowledgements

The author is gratefully indebted to the Ametek Computer Research Division of Arcadia, California for answering many questions and providing access to one of their System 14 hypercubes and to Ncube for providing access to their 8 MHz/512K hypercube.

References

- [1] Ametek Computer Research Division, *Ametek System 14 User's Guide*, Ametek V12970, Arcadia, CA, May, 1986.
- [2] H. J. Curnow and B. A. Wichman, *A Synthetic benchmark*, Computer Journal, **19** (1976).
- [3] T. H. Dunigan, *Hypercube Performance*, Hypercube Multiprocessors 1987, ed. M. T. Heath, SIAM, Philadelphia, 1987.
- [4] T. H. Dunigan, *A Message-passing Multiprocessor Simulator*, Tech. Rept. ORNL/TM-9966, Oak Ridge National Laboratory, Oak Ridge, TN (1986).
- [5] G. C. Fox and S. W. Otto, *Algorithms for concurrent processors*, Physics Today, May 1984, pp. 50-59.
- [6] Dirk C. Grunwald and Daniel A. Reed, *Benchmarking hypercube hardware and software*, Hypercube Multiprocessors 1987, ed. M. T. Heath, SIAM, Philadelphia, 1987.
- [7] Intel, *iPSC User's Guide*, Intel 17455-03, Portland, Oregon, October, 1985.
- [8] A. Kolawa and S. Otto, *Performance of the Mark II and Intel Hypercubes*, Hypercube Multiprocessors 1986, ed. M. T. Heath, SIAM, Philadelphia, 1986, pp. 272-275.
- [9] Ncube, *Ncube Handbook*, Ncube V1.1, Beaverton, OR, 1986.
- [10] C. L. Seitz, *The cosmic cube*, Comm. ACM, **28** (1985), pp. 22-33.
- [11] R. Weicker, *Dhrystone: a synthetic systems programming benchmark*, Comm. ACM, **27** (1984), pp. 1013-1030.

**DISTRIBUTION OF
ORNL/TM-10400**

- | | | | |
|--------|--|--------|--|
| 1. | J. Barhen | 23. | J. Wooten |
| 2-6. | T. H. Dunigan | 24. | A. Zucker |
| 7. | Y. H. Etheridge | 25. | Central Research Library |
| 8-9. | R. F. Harbison/
Mathematical Sciences Library | 26. | K-25 Plant Library |
| 10. | G. A. Geist | 27. | ORNL Patent Office |
| 11. | J. A. George | 28. | Y-12 Technical Library
Document Reference Section |
| 12. | M. T. Heath | 29. | Laboratory Records Department--RC |
| 13. | J. K. Ingersoll | 30-31. | Laboratory Records Department |
| 14. | J. M. Jansen | 32. | P. W. Dickson, Jr. (Consultant) |
| 15. | F. C. Maienschein | 33. | G. H. Golub (Consultant) |
| 16. | E. Ng | 34. | R. W. Haralick (Consultant) |
| 17. | C. H. Romine | 35. | D. Steiner (Consultant) |
| 18-22. | R. C. Ward | | |

EXTERNAL DISTRIBUTION

36. Dr. Donald M. Austin, Office of Scientific Computing, Office of Energy Research, ER-7, Germantown Building, U.S. Department of Energy, Washington, DC 20545
37. Dr. Bill L. Buzbee, C-3, Applications Support & Research, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
38. Dr. John Cavallini, ER-7, GTN, Office of Scientific Computing, Department of Energy, Washington, D.C. 20545
39. Dr. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550
40. Dr. James Coronas, Ames Laboratory, Iowa State University, Ames, IA 50011
41. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303
42. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
43. Prof. Geoffrey C. Fox, Associate Provost for Computing, California Institute of Technology, Pasadena, CA 91125
44. Dr. R. E. Funderlic, Dept. of Computer Science--Box 8206, North Carolina State University, Raleigh, NC 27695-8206
45. Dr. Robert Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
46. Prof. Malvyn Kalos, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10013
47. Dr. Jeff Kien, Ametek Corp., 610 North Santa Anita Avenue, Arcadia, CA 91006
48. Prof. David J. Kuck, Center for Supercomputing R&O, University of Illinois, 1384 W. Springfield Avenue, Urbana, IL 61801

49. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
 50. Dr. Paul C. Messina, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
 51. Dr. Cleve Moler, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
 52. Dr. James M. Ortega, Department of Applied Mathematics, University of Virginia, Charlottesville, VA 22903
 53. Dr. John F. Palmer, NCUBE Corporation, 915 E. LaVieve Lane, Tempe, AZ 85284
 54. Dr. Jesse Poore, Computer Science Dept., 107 Ayres Hall University of Tennessee Knoxville, TN 37996
 55. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550
 56. Capt. John P. Thomas, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
 57. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
 58. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37830
- 59-89. Technical Information Center.