NASA Technical Memorandum 101952

A Message Passing Kernel for the Hypercluster Parallel Processing Test Bed

Richard A. Blech
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio

Angela Quealy
Sverdrup Technology, Inc.
NASA Lewis Research Center Group
Cleveland, Ohio

and

Gary L. Cole
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio

Prepared for the
Fourth Conference on Hypercubes, Concurrent Computers, and Applications cosponsored by the U.S. Department of Energy (Applied Mathematical Sciences Program), Strategic Defense Initiative Organization (Office of Innovative Science and Technology), Joint Tactical Fusion Program Office, U.S. Air Force (Electronic Systems Division), Air Force Office of Scientific Research, and NASA Ames Research Center
Monterey, California, March 6-8, 1989

NASA

A MESSAGE-PASSING KERNEL FOR THE HYPERCLUSTER PARALLEL-PROCESSING TEST BED

Richard A. Blech
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

Angela Quealy
Sverdrup Technology, Inc.
NASA Lewis Research Center Group
Cleveland, Ohio 44135

and

Gary L. Cole
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

ABSTRACT

A Message-Passing Kernel (MPK) for the Hypercluster parallel-processing test bed is described. The Hypercluster is being developed at the NASA Lewis Research Center to support investigations of parallel algorithms and architectures for computational fluid and structural mechanics applications. The Hypercluster resembles the hypercube architecture except that each node consists of multiple processors communicating through shared memory. The MPK efficiently routes information through the Hypercluster, using a message-passing protocol when necessary and faster shared-memory communication whenever possible. The MPK also interfaces all of the processors with the Hypercluster operating system (HYCLOPS), which runs on a Front-End Processor (FEP). This approach distributes many of the I/O tasks to the Hypercluster processors and eliminates the need for a separate I/O support program on the FEP.

INTRODUCTION

Two research areas which are critical to the future progress of aerospace technology are computational fluid mechanics (CFM) and computational structural mechanics (CSM). The practical limits of applications in both of these areas are being set by the state-of-the-art in computer architecture and software techniques. Parallel processing is an architectural concept which has the potential for vastly improving the performance of future computer systems. However, the use of parallel-processing architectures will require a reassessment of numerical methods and software techniques that are currently being used on single-processor computers for CFM/CSM. Likewise, CFM/CSM requirements will, more than likely, impact the directions taken in the development of future parallel architectures.

There have been many parallel architectures proposed in the last several years (Refs. 1 to 3). Several have even enjoyed commercial success (Ref. 4). These architectures basically fall into two categories: shared memory and distributed memory. Each category has its own advantages and disadvantages, as shown in Table I. Generally, distributed-memory architectures are characterized by a larger number of processors and slower processor-to-processor communications than their shared-memory counterparts. This results in potentially high performance for problems that exhibit fine-grain parallelism. That, and relatively low cost, have combined to make distributed-memory architectures the focus of considerable research. The promise of higher performance, however, comes at the price of additional effort which must be put into the programming and mapping of applications onto this architecture. In general, the choice of an architecture for a given application is not clear-cut. Future technological breakthroughs may tip the scales in favor of one over another.

Researchers at NASA Lewis have initiated the development of the Hypercluster, a parallel-processing test bed which combines shared and distributed-memory architectures. This allows, within a common programming and operating evironment, investigations into the relative merits of both architectures and a platform for investigating hybrid (combined shared and distributed memory) architectures. The Hypercluster also provides the capability to develop and evaluate programming and operating software for distributed, shared and hybrid-memory architectures.

The success of the Hypercluster will depend largely on the software tools provided with it. An Initial Operating Capability (IOC) has been defined which will provide the programming and operating tools necessary to use the Hypercluster (Ref. 5). This includes a FORTRAN 77 compiler for application program development, a library of subroutines to support vector and parallel processing, and an interactive operating system. Future enhancements to the Hypercluster software environment will further ease the application development task for hypercube-like and other architectures.

An integral part of the IOC is the Hypercluster Message-Passing Kernel (MPK) which resides on each of the processors. The MPK provides basic interprocessor communication and synchronization services as well as an interface to the Hypercluster Operating System, HYCLOPS. The Hypercluster MPK is unique from other hypercube kernels in that it deals with multiprocessor shared-memory nodes and has a shared-memory interface to HYCLOPS. A description of the MPK and how it interacts with the Hypercluster architecture is given in this report.

HARDWARE OVERVIEW

The architecture for a four-node Hypercluster prototype is shown in Fig. 1. A four-node Hypercluster consists of four clusters of processors interconnected in a binary n-cube configuration by internode links. The links allow communication between nodes and consist of two Control Processors (CPs) communicating through dual-ported memory. Each node can have a special purpose link dedicated to communicating with a Front-End Processor (FEP). The architecture of these links is identical to the architecture of the internode links. The FEP and the associated CPs on its bus can be considered as a special-purpose Hypercluster node. Thus, if desired, FEP functions can take advantage of parallel processing. The Hypercluster user interacts with the hardware through HYCLOPS, which runs on the FEP.

Each node of the Hypercluster can consist of any number and combination of processors. The types of processors currently used in the Hypercluster are scalar processors (SPs) and vector processors (VPs). The VPs act as peripherals to the scalar processors, so that every node containing a VP must have an SP dedicated to operating the VPs. Processors within a node communicate through shared memory. The memory may be dual-ported memory located on the processor board itself, or a separate memory board.

The VPs and SPs perform the program computations within the Hypercluster. The CPs can also perform computations, but this could degrade their performance as communication processors. For this reason, CPs are usually dedicated to communication only. However, applications requiring only nearest-neighbor communications may be able to take advantage of the fast dual-port memory interface between the CPs. This approach would use the Hypercluster as a ring architecture, where each node in the ring communicates with its neighbors through dual-ported memory.

Within each communication link, each processor has the capability of interrupting the other processor, informing it that information is waiting within the dual-port memory. The interrupted processor can then take this information and transfer it to an SP within it's node, or to another CP if the information is destined for another node. It is important to note that the CPs can route information throughout the Hypercluster without interrupting or requiring the assistance of the SP's or VP's, which are busy computing the current application.

The Hypercluster resembles other hypercube architectures in many respects, but the following summarizes important differences:

(1) Each node of the Hypercluster can consist of more than one scalar and/or vector processor.

(2) Processors within a node communicate through shared memory.

(3) Independent processors (CPs) perform the internode communication without interrupting the processors within a node that are performing the application computations. These CPs are programmable, allowing investigation of various message-passing protocols.

(4) There can be more than one communication link between the FEP and the Hypercluster nodes. These links are intelligent, allowing experimentation with distributed operating-system concepts.

(5) The processor technology within each node is not limited to one particular vendor. The use of a standard bus allows any processor board available for that bus to be incorporated. This feature also provides a rapid method for upgrading the system hardware.

MPK DESIGN AND STRUCTURE

The Hypercluster MPK is intended to provide basic operating and communication functions to the various processors. The communication services of the MPK can be accessed from a FORTRAN program through subroutine calls. This is done by providing a library of FORTRAN-callable routines which activate the MPK through a software interrupt. The kernel also allows the FEP to control and communicate with the Hypercluster processors. A message-passing protocol defines the way in which all Hypercluster processors and the FEP communicate. Messages can be categorized as follows:

(1) Data to be transferred between processors

(2) Control information from the FEP

(3) Error or status information from the processors

(4) Data destined for the FEP for analysis or storage

The first category consists of messages to support the exchange of data between Hypercluster processors. This includes CPs, SPs, and the FEP. Messages in the second category are used mainly by HYCLOPS to control the operation of the Hypercluster. Messages to start, stop and resume execution are included here, as well as timer support. The third category encompasses messages from the Hypercluster regarding errors. These messages usually contain processor diagnostic and status information at the time of the error, and are sent to the FEP for processing by HYCLOPS. The last message category is used to support I/O for the Hypercluster processors. Currently, all I/O is provided through the resources of the FEP. Messages to store data on disk (from a FORTRAN WRITE statement, for example) or to display graphical information are sent to the FEP, which processes them.

The MPK runs on each of the Hypercluster processors. A "layered" approach is used to define the various kernel elements as shown in Fig. 2. The hardware is initialized and tested by the INIT layer of the kernel, which resides in ROM. INIT tests the Hypercluster processors and communication links. It then loads the outer layers of the MPK, which are sent by HYCLOPS. HYCLOPS maintains a configuration file for the current Hypercluster hardware. The configuration file tells HYCLOPS how many nodes and processors to load, and what hardware specific information to initialize for the MPK. Loading the MPK, as opposed to having it ROM resident, aids in the development and debugging process, since changes can be made to the MPK and quickly tested.

The INIT layer is followed by the Link Transfer Utility (LTU), which performs the most basic data transfer operation across a Hypercluster internode link. The LTU moves a message to/from a specified buffer in a Hypercluster node from/to the dual-ported memory shared by the CPs. One CP in a link informs the other CP in the link of a message transfer either by a flag or an interrupt. The LTU in the receiving CP then sets up the information needed by the outer MPK layers to further process the message.

One of the objectives of the MPK design is hardware retargettability, which is accommodated by the lowest two layers of the MPK. The INIT and LTU layers contain all the hardware dependent code associated with the processors and their communication links. These layers contain the only code that would have to be changed if different processor boards or communication links were used. The INIT and LTU are coded in assembly language for efficiency. The remaining software layers are also coded in assembly language. This ties the upper layers of the MPK to one microprocessor family, but not to any particular board-level product based on that family. It was felt that this approach provided sufficient flexibility and speed for the initial version of the MPK and the prototype Hypercluster hardware. The use of the language C provides hardware independence across several microprocessor families, but at a performance cost. It will be investigated for coding future versions of the MPK.

The remaining layers of the Hypercluster kernel do not require specific information about the underlying hardware. The Network Communication Handler (NCH) is the next of these layers, and is responsible for the routing of messages within the Hypercluster. The NCH selects the appropiate link (and as a result, the CP) to send a message through based on the destination address. The destination address consists of a node and processor identification (ID). The NCH routes the message so as to minimize the total path length that it travels to reach its destination node. It then selects the appropriate processor within that node to receive the message.

The message transmission/reception layer takes action on a message that is received or is to be sent. When sending a message, this layer will acquire any necessary buffers, copy the message and set queue entries in the required processor as directed by the NCH. If a message is received, it is decoded and the appropriate handling routine is called. Some messages, destined for an SP in a node, are handled by the CPs without SP intervention. For example, a message to read the memory of an SP would be handled by one of the CPs in the node. The CP would read the requested data from the SP through its dual-port memory on the node bus. It would then form a response message with the data and send this message to the requester.

The operating system and high-level language interface layers give HYCLOPS and the application program access to the services provided by the MPK. HYCLOPS runs on the FEP, which has a unique node and processor ID. This allows HYCLOPS to send and receive messages to/from any of the Hypercluster processors. The Hypercluster CPs on the FEP bus contain a special version of the MPK which implements a shared-memory interface through which messages are exchanged with the FEP. I/O and control functions are built into HYCLOPS and the MPK's operating-system interface layer. This eliminates the need for a separate host program on the FEP to handle these functions. Another unique feature of this layer is the capability to read or write processor memory "on the fly". This means that HYCLOPS can read or change program variables on the Hypercluster processors while the application program is running.

The high-level language interface allows a user's application program to call various MPK services. This is accomplished through a software interrupt, or trap, which is issued from a FORTRAN subroutine. A pointer passed in an address register indicates where the message is that the MPK must service. A collection of subroutines called the parallel-processing support library issue various messages in this manner, giving the programmer high-level access to the MPK. Application programs can potentially issue messages from any of the categories described earlier, allowing control of data transfers or even system operation from any Hypercluster processor.

IMPLEMENTATION DETAILS

The basic unit of information which is transferred between Hypercluster processors and/or the FEP is a message. A message consists of a header and an optional message text. The format of the message header is as follows:

Offset (bytes)	Description
0	Number of bytes in message
2	Message type
4	Source node and processor ID
6	Destination node and processor ID
8	Destination address
12	Message text pointer

The first field in the header defines the number of bytes in the message, not including the header. The maximum message size is limited to 64KB since this field is 16 b wide. The message type field identifies what action is to be taken in response to this message. This is followed by 16-b codes for both the source and destination for this message. Each of these codes consists of an 8-b node and an 8-b processor ID. Every processor in a node, including the CPs, has a unique processor ID. The processor ID represents an entry into a table containing the base addresses of every processor's dual-port memory on the node bus. This base address, combined with the destination address in the fifth field, yields a unique address within the shared-memory node environment. The final field in the header is a pointer to the optional message text. This allows additional message data to be located anywhere in a Hypercluster node's memory. The total memory required for the header is 16 B.

Message Queues

Each processor in the Hypercluster maintains a message queue to receive all incoming messages. A message queue consists of a number of 32-b slots, each of which is a pointer to a message header. After the initialization and self-test routines are performed, each processor goes into a polling mode, waiting for messages in its queue. The presence of a message in the queue is indicated by a nonzero value at the current queue service pointer. The queue service pointer (QSP) is maintained by the processor to keep track of which message in the queue it is currently servicing. When a message is serviced, the value at the QSP is zeroed out. This indicates that the slot is again available to hold message pointers.

Queues are serviced first-in-first-out (FIFO). A queue entry pointer (QEP) is maintained to indicate the next free queue slot. Any processor wanting to send a message to another processor must first access the QEP to see where in the receiving processor's queue it must place the message pointer. If the QEP is at the last queue element, the new message pointer is entered at this location, and the QEP is reset to the beginning of the queue. If this queue slot is not empty, a queue overflow message is issued to the user, and the Hypercluster is halted.

The QSP is managed by the receiving processor only. The QEP is controlled by the sending processor (through dual-port memory). Since the QEP can be changed by any of several processors, it must be treated as a shared resource. Thus a multiprocessor primitive that assures mutual exclusion must be used to access the QEP. This can be done using a test-and-set (TAS) instruction.

Message Size and Buffering

The values in each slot of the message queue point to the header of the message. The header may contain enough information itself, or may also contain a pointer to a message text area, where further information is stored. When a processor in the Hypercluster wishes to send a message to another processor, it must form the message (a header and, if necessary, a text area) somewhere in memory. The message text can be buffered or unbuffered at the option of the programmer. An example of an unbuffered message would be one where the text pointer in the header is pointing directly to a FORTRAN array. With a buffered message, this array would be copied to a general buffer area. The unbuffered message eliminates the overhead of copying the message text to another buffer area, and is thus more effecient. However, the unbuffered message requires that the memory space occupied by the message text not be altered until the message has been completely serviced by the MPK.

Messages can be of any length, but the NCH breaks large messages into segments of a more managable size. The NCH will check the message header for the number of bytes in a message. If the size is larger than 512 B, the message is split up and sent in 512-B segments. A buffer area of 128K is maintained in each processor for messages, and is treated as consisting of many fixed-size segments. Each segment is large enough to hold one 16-B header plus 512 B of data. This allows the buffer area to be treated as a ring buffer. The fixed-segment ring buffer approach, being simple to implement, is more time efficient than other approaches at the cost of memory efficiency. Since each Hypercluster processor has a large local memory, this tradeoff was felt to be reasonable.

A message buffer pointer (MBP) is maintained to indicate to the sending processor where in the 128K buffer the next message can be placed. The MBP is a shared resource (among processors in a node) and as such must be accessed through a TAS instruction. Before writing into the buffer, a processor must verify that the "message type" location in the header is zero. This indicates that this area of the buffer is free. If it is not zero, a buffer overflow message is issued. Otherwise, the header and the message are written into the buffer, and the MBP is incremented by 528. If the new MBP value is beyond the buffer area, it is wrapped around to the beginning.

Message Broadcasting

There are some instances where a processor or the FEP may want to send a message to all processors in the Hypercluster. This is called broadcasting. A broadcast message is indicated by using a unique destination processor ID. The destination node ID is not used. Messages can be broadcast to CPs only, SPs only, or to all processors. For example, one may want to issue control information to CPs only, and data to SPs only. When a CP receives a broadcast message, it keeps a copy of the message for itself, and sends the message on to another CP as determined by an embedded spanning tree architecture (Ref. 6). The CP then takes the appropriate action defined by the message type for each processor in its node. For example, a data broadcast message to write information to all SPs would be intercepted by a CP in each node, which would write the data to each SP within it's node. Only data transfer and control messages support broadcasting.

Direct Data Transfer

As described earilier, the MPK supports several categories of messages. The first category, data transfer, warrants further discussion here because of several unique characteristics. Data transfer messages in the MPK can be direct and indirect. The indirect transfer is typical of the "SEND" message supported on many hypercube systems. The message buffer address and destination node are specified, but the address of the destination variable (in the application program) is not. This link is made through a "RECEIVE" call on the destination node, where the address of the receiving variable is specified as an argument. The "RECEIVE" message causes data to be transferred from a general buffer area to a specific variable area. The buffering overhead and the overhead associated with a "RECEIVE" call can be avoided through a direct form of data transfer message. Here, the address of the actual receiving variable is specified in the "SEND" message. Synchronization between the source and destination nodes can be accomplished through a flag variable included in the message, without the need for a "RECEIVE" call on the destination node.

The direct data transfer message is more efficient than the indirect form, but puts the burden on the programmer of knowing the physical address of the destination variable. This can be accomplished, for example, with FORTRAN common blocks if the compiler's common block allocation method is known and the common block addresses are known or can be specified. Identical common blocks are declared on the sending and receving processors, resulting in identical addresses for variables within these common blocks. The names of variables within these common blocks can then be used for the destination variable argument in a direct data transfer message call.

PERFORMANCE TESTS AND RESULTS

Several communication performance tests were run on the Hypercluster with version 1.0 of the MPK. The objectives of these tests were to verify that the MPK: (1) was functioning correctly and (2) provided realistic node to node communication performance. Meeting these objectives assures that results generated on the Hypercluster can be related to those generated on other hypercube systems, and that the architectural flexibility of the Hypercluster can be effectively utilized.

The first test was a node to node communication test. In this test, an SP in one node issues a message to transfer a specified number of bytes to an SP in the other node. This process is repeated in a loop for a specified number of times. An average node to node communication rate as a function of the number of bytes transferred can be determined from this test. Figure 3 shows a plot of communication rate as a funtion of the number of bytes transferred. The data for the Intel iPSC and the NCUBE systems were taken from (Ref. 7). For messages less that 512 B, the Hypercluster outperforms both commercial systems. Beyond 512 B, the Hypercluster link performance saturates at about 300 KB/sec. This is due to the packetization of messages greater than 512 B. The packet size can be varied, and this option will be studied in the future.

It should be noted that this test was run SP to SP. This means that the CPs in the node had to transfer message information across the node bus, adding considerable overhead. Although the CPs are not as computationally powerful as the SPs or VPs, there may be applications that can benefit from running on the CPs themselves. In this case, transfers between CP and SP would be unneccessary, and node to node communication performance would be improved considerably.

Another test which is commonly used to measure communication performance is the ring test, as described in (Ref. 8). In this case, the Hypercluster was mapped into a 4 node ring. From this test, the performance parameter, tcomm, can be determined, where tcomm is defined as the transfer time in microseconds for a 32-b word. The results for this test are shown in Table II, along with results from other hypercubes reported in (Refs. 8 and 9). As shown in Table II, the MPK results compare favorably with the other systems, verifying that the Hypercluster MPK provides realistic performance for message passing operations. Note that the NCUBE figures are given for 2 versions of their node operating system, standard and extended VERTEX. The results shown previously in Fig. 3, as taken from (Ref. 7), were probably for standard VERTEX, although this was not mentioned in (Ref. 7). The use of extended VERTEX would probably result in better performance for NCUBE (in Fig. 3) with smaller messages.

All of the results shown were generated with version 1.0 of the MPK and prototype hardware. The message packet size used with version 1.0 is currently 512 B. This size can be changed, depending on the average message size required for a given application. A larger packet size will result in better performance for larger messages. Code optimization of the MPK will also improve performance. In addition, the CPs in the Hypercluster are 10 MHz processors with a 16-b bus interface. Communication performance could be further improved by using faster processors with a 32-b bus interface. These options will be pursued for future versions of the Hypercluster.

CONCLUSIONS

Future plans call for the investigation of several fluid-flow problems using the Hypercluster prototype and subsequent versions of the Hypercluster. The results shown here demonstrate that the Hypercluster test-bed and the MPK provide an efficient

CONCLUSIONS

Future plans call for the investigation of several fluid-flow problems using the Hypercluster prototype and subsequent versions of the Hypercluster. The results shown here demonstrate that the Hypercluster test-bed and the MPK provide an efficient
distributed-memory communication mechanism that can be used in these investigations. In addition, the multiple-processor, shared-memory clusters at each node make the Hypercluster an attractive and powerful tool for investigating the relative merits of shared, distributed, or hybrid architectures for computational fluid and structural mechanics.

REFERENCES

1. K. Hwang, Multiprocessor Supercomputers for Scientific/Engineering Applications, Computer, 18[6] (1985) pp. 57-73.

2. K. Hwang, Tutorial Supercomputers: Design and Applications, IEEE Computer Society Press, Los Angeles, 1984.

3. P.B. Schneck, D. Austin, S.L. Squires, J. Lehmann, D. Mizell, and K. Wallgren, Parallel Processor Programs in the Federal Government, Computer, Computer, 18[6] (1985) pp. 43-55.

4. N. Mokhoff, Parallelism Breeds a New Class of Supercomputers, Computer Design, 26[6] (1987) pp. 53-60,62,64.

5. G.L. Cole, R. Blech, and A. Quealy, Initial Operating Capability for the Hypercluster Parallel-Processing Test Bed, NASA TM-101953, 1989. (Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications, Mar. 6-8, 1989, SIAM, Philadelphia, PA, to be published.)

6. J.E. Brandenburg, and D.S. Scott, Embeddings of Communication Trees and Grids into Hypercubes, iPSC Technical Report No. 1, Intel Corp., Beaverton, OR, 1986.

7. T.H. Dunigan, Hypercube Performance, Hypercube Multiprocessors 1987, M.T. Heath, ed., SIAM, Philadelphia, PA, 1987, pp. 178-192.

8. A. Kolawa, and S.W. Otto, Performance of the Mark II and Intel Hypercubes, Hypercube Multiprocessors 1986, M.T. Heath, ed., SIAM, Philadelphia, PA, 1986, pp. 272-275.

9. T.N. Mudge, G.D. Buzzard, and T.S. Abdelrahman, A High Performance Operating System for the NCUBE, Hypercube Multiprocessors 1987, M.T. Heath, ed., SIAM, Philadelphia, PA, 1987, pp. 90-99.

TABLE I. - SHARED VERSUS DISTRIBUTED MEMORY

Architecture	Number processors	Media speed	Software overhead	Programming
Shared	Few	Fast	Low	Simple
Distributed	Many	Slow	High	Complex

TABLE II. - VALUES OF TCOMM (μsec) FOR THE HYPERCLUSTER AND OTHER HYPERCUBE SYSTEMS

Number of bytes	Hypercluster MPK V1.0	NCUBE standard Vertex	NCUBE ext. Vertex	IPSC CrOS	IPSC IHOS	Mark II CrOS
8	126.5	245.76	46.76	160	5960	86.0
64	25.3	41.79	10.37	80	777	45.5
256	14.6	19.88	6.45	79	202	41.4

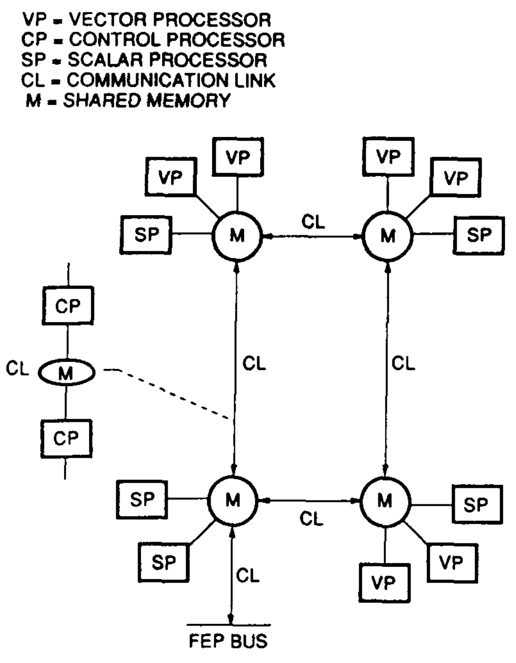

Figure 1. - Hypercluster test bed architecture.

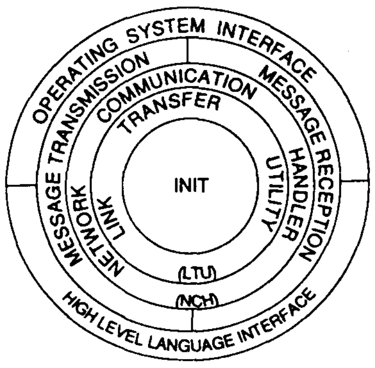
OPERATING SYSTEM INTERFACE
MESSAGE TRANSMISSION
COMMUNICATION TRANSFER
MESSAGE RECEPTION HANDLER
UTILITY
INIT
NETWORK LINK
(LTU)
(NCH)
HIGH LEVEL LANGUAGE INTERFACE

Figure 2. - The hypercluster MPK.

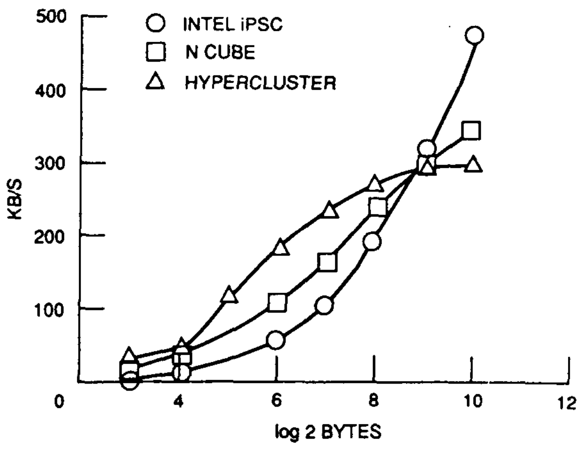

Figure 3. - Hypercluster node to node performance. (Data from Dunigan, Hypercube multiprocessors, 1987)

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-101952	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Message Passing Kernel for the Hypercluster Parallel Processing Test Bed		5. Report Date
		6. Performing Organization Code
7. Author(s) Richard A. Blech, Angela Quealy, and Gary L. Cole		8. Performing Organization Report No. E-4652
		10. Work Unit No. 505-62-21
9. Performing Organization Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546-0001		14. Sponsoring Agency Code

15. Supplementary Notes

Prepared for the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, cosponsored by the U.S. Department of Energy (Applied Mathematical Sciences Program), Strategic Defense Initiative Organization (Office of Innovative Science and Technology), Joint Tactical Fusion Program Office, U.S. Air Force (Electronic Systems Division), Air Force Office of Scientific Research, and NASA Ames Research Center, Monterey, California, March 6-8, 1989. Richard A. Blech and Gary L. Cole, NASA Lewis Research Center; Angela Quealy, Sverdrup Technology, Inc., NASA Lewis Research Center Group, Cleveland, Ohio 44135.

16. Abstract

A Message-Passing Kernel (MPK) for the Hypercluster parallel-processing test bed is described. The Hypercluster is being developed at the NASA Lewis Research Center to support investigations of parallel algorithms and architectures for computational fluid and structural mechanics applications. The Hypercluster resembles the hypercube architecture except that each node consists of multiple processors communicating through shared memory. The MPK efficiently routes information through the Hypercluster, using a message-passing protocol when necessary and faster shared-memory communication whenever possible. The MPK also interfaces all of the processors with the Hypercluster operating system (HYCLOPS), which runs on a Front-End Processor (FEP). This approach distributes many of the I/O tasks to the Hypercluster processors and eliminates the need for a separate I/O support program on the FEP.

17. Key Words (Suggested by Author(s)) Hypercube Parallel processing Software Fluid mechanics	18. Distribution Statement Unclassified – Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	10	A02

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161