A DISTRIBUTED HYPERCUBE FILE SYSTEM

Robert J. Flynn and Haldun Hadimioglu

Electrical Engineering and Computer Science Department
Polytechnic University
333 Jay Street
Brooklyn, New York 11201

Abstract. *For the hypercube, an autonomous physically interconnected file system is proposed. The resulting distributed file system consists of an I/O organization and a software interface. The system is loosely-coupled architecturally but from operating systems point of view a tightly-coupled system is formed in which interprocessor messages are handled differently from file accesses. A matrix multiplication algorithm is given to show how the distributed file system is utilized.*

1 Introduction

Concurrent computer systems are typically applied to very large, partitionable computationally demanding scientific problems. These problems are very large either because a large number of computations (work) are done for each data element or because there is a very large number of data elements (volume) or both. This usually results in large, frequent data movements, increasing the I/O load. Dividing the problem among a large number of processors can clearly cut the amount of (work) time required to do a problem. Partitioning a problem (volume) among many computing elements can add I/O to an already large data volume problem.

Hypercube systems represent a particular form of interconnected homogeneous machines [1], [2]. The principal advantage is in its simple, short, message passing capability, taken together with an easily scalable form. While it is clear that work time can be cut and that one can deal with larger work problems, the issue of data volume and I/O remain.

The decomposition of the work required in the hypercube form can be independent of the logical decomposition of the data volume. If the data volume is memory bound an artificial repartition of the process space is required. For this reason, the logical introduction of a distributed file system is required. The topology of a distributed file system can be independent of the process distribution topology. Here, however, an embedded, autonomous sub-hypercube disk-based file system is proposed. The system does not replace message based transfers among processors. It supplements them with distributed file transfers.

Recently, the I/O problem has been considered and termed as the concurrent I/O problem [3]. A paper at the third hypercube conference [4] proposes a system similar tc the system, independently developed and proposed in this paper. Witkowski et al suggest a Concurrent I/O (CIO) system and a hypercube Concurrent File System (CFS) for high bandwidth concurrent I/O. A CIO node connects a subcube of processing nodes to the outside world. A number of CIO nodes are interconnected in a hypercube fashion.

Commercial hypercube manufacturers are following the same path to have more I/O utilization. The NCUBE NChannel™ provides high speed parallel I/O [5]. I/O

channels connect the hypercube array to a number of NChannel I/O boards. Each I/O channel supports a single disk controller. The disk system software running on each I/O board forms a hierarchical file system and a "disk farm" can be configured.

Intel iPSC/2 has one communication link unused per node for I/O other than interprocessor communication I/O [6]. It also allows a high-speed iLBX™ interface board to be attached to each node.

The FPS Mark II T Series hypercubes have a system board on the system ring [7]. The board has a disk which can hold data for computation.

2 Hypercube Algorithms

CALTECH experience with hypercubes has pointed out four factors for efficient concurrent computation [8]. First, communication time especially due to hardware restrictions must be reduced. Direct-Connect™ routing employed on iPSC/2 aims at this point [9]. Second, induced parallel overhead due to synchronization, new programming language and operating systems constructs must be low. Third, the load balance of the application should not be too difficult to maintain during the computation. Fourth, a low ratio of the communication to the computation is a necessity for high efficiencies [8]. Side effects of these are that communication and concurrency compete with each other and load balancing and communication compete with each other.

Hypercube applications have the properties of high concurrency, natural load balance, a regular communication graph, little communication and a large data space. Each data element is reused many times i.e. many accesses to each individual data element are made. This coupled with the fact that many calculations are done per data element results in a large number of computations for the application.

The computation pattern of the applications makes use of chunks of sequential data elements at any time, without imposing difficulties and changes on the communication pattern of the application. Many large strips of data move simultaneously the net effect of which is that there is very large, coarse grain, data flow in the system at any moment.

The domain decomposition phase of algorithm development determines how the communication graph of the algorithm matches the hypercube network, the load balance and the concurrency. It is because of these reasons that decomposition has become a frequently discussed subject in hypercube literature. It is the aim of this paper to show that an optimum decomposition is not enough and current hypercubes can be more efficient and more powerful if they are provided with a distributed file system.

3 The Need for more I/O

The nature of hypercube processing is that each node can independently have I/O. I/O bandwidth in hypercubes is ideally proportional to the number of nodes. Thus, hypercubes offer higher I/O capability than serial computers and some other parallel computers. The scalability property of hypercubes makes them suitable for larger problems that as new nodes are added the memory size increases linearly and communication time is not affected as much. But for hypercubes the I/O potential must be used efficiently in order to have a real computing power.

One could add more and more memory at each node in order to solve some of the I/O issues. But, this is not an optimum solution since it is possible to find larger and larger applications for which the hypercube is too small. Even if these larger applications can be run, it will be at lower speeds. The reason is that still these large memories are not used efficiently. Long messages are needed for reading and writing of these large memories.

At some point the logical memory hierarchy of file is both appropriate to and needed by hypercube systems. That is, at some point and with some problems, one must deal with files and large disk systems. Current systems treat a hypercube system as a backend processor with all of the basic file management work being done by a single front end computer. A large disk attached to a single front end forms a type of single point global file system. The centralized disk reduces the problem to the one seen in shared memory systems: A single point possible bottleneck is created in the context of a system designed to distribute work.

The issue of file systems is seen principally at startup time, when large amounts of data are scattered to appropriate processors. Application programs are written such that all data is sent to the Processing Nodes (PNs) before PNs actually start computation. After that, interprocess communications are treated as message passing. When access to a file has to be made, one invokes the

front end processor. To the degree that a conceptual piece of a file exists in the memory of different processors one can move them about as if they are very long messages. This causes long startup delays and ultimately limits the scalability of the machine form.

In order to solve the basic physical I/O problem, one could attach a small disk to each processor. This, however, is expensive and not scalable. It does not take into account the need for different clusters of processors to cooperatively work on similar pieces or phases of related data objects.

Scaling up a hypercube for higher speeds has been possible as noted in the literature. Such universal programs [10] do not, however, reflect the time to distribute data from the intermediate host (IH) to the PNs and the time to gather results from the PNs to the IH. These programs are that before the computation starts the whole data domain is partitioned. After the start of the execution, there is virtually no new data distribution from the IH. Such a scheme certainly sets a limit on the size of applications based on the size of each local memory.

During a hypercube program execution each PN gets an equal part of the domain for load balancing purposes. Some PNs may need some part(s) of the initially distributed data volume that are stored on other PNs. Some PNs may need (intermediate) results produced by other PNs. These large data block transfers are made by sending messages through PN-PN links.

In conventional hypercube systems there is little notion of files for the processing nodes. Messages are treated like files. One must deal with files. Messages are not files. The I/O problem is worsened by the fact that using more PNs results in more communication of the type mentioned above. This is because the more partitioning of a problem, the larger the data flow, i.e. more long messages have to be sent. Now the system is such that a) There are long startup delays. b) There are long communication times. c) The speed is slower. d) There is a low throughput. e) The system is not scalable. f) The efficiency is lower. g) Programs are not universal.

4 The Distributed File System

We suggest a distributed file system, one in which large volumes are distributed to several disks, interconnected to each other and at the same time with specific disks more tightly coupled to clusters of processors, or pieces of the hypercube.

The file concept is important in concurrent systems as it is in sequential, array and pipelined computers. A file is not more memory. A file is different from a message. A file is different from a disk as much as a memory can be distinguished from a disk. Files cannot be treated the way variables contained in memories are treated.

The I/O organization and the software interface are the two aspects of this research. The distributed hypercube file system must be designed so that

i) The whole system is scalable.
ii) The concurrency is not lowered.
iii) The communication is not increased.
iv) Load balancing is not disturbed.

A schematic description of the I/O organization is given in figure 1. A number of sets of disks are interconnected. Each set of disks is connected to a cluster of PNs. Each set of disks is controlled by dedicated hardware: a Disk Node (DN). The host is connected to all DNs.

The I/O organization and the software interface that supports the I/O organization are such that files which are large data volumes are distributed across the DNs. Each DN has a number of blocks of a file. Large data transfers are made in file blocks. Block transfers involve PN-DN and DN-DN links. The interprocessor links, PN-PN links, are not used for file block transfers. At any time any block of any file can be accessed by any of the PNs, the DNs or the IH. The system makes file sharing easier especially among the PNs.

The following points are considered for satisfying the above four requirements:

a) DNs should have a low degree. Their interconnection must be simple.
b) Maximum distance between any two DNs must be small.
c) Routing among the DNs must be simple and must not require global information.
d) Each DN must be simple and possibly identical to form another set of homogeneous elements in the system.
e) The distance between any pair of a PN and a DN must be small.

DNs are not passive or slave when compared with PNs. Their software includes the software interface of the distributed file system. They can make decisions for efficient use of files. The software interface is basically concerned with

buffer management, naming and directory management. It is implemented as a file service which consists of a number of file servers each one of which runs on a DN and independent of other file servers. File servers are responsible for coherence, contention, locking of blocks (and files if necessary) and synchronization. A file server can locate any block of any file and access it.

The efficiency of the file system is affected by the way disk node buffers are managed. Each of these buffers occupies most of the local memory of a DN. Keeping most recently used file blocks in the buffer is important for file sharing. Sometimes a PN, PNx, can request a block on another DN, DNb. The closest DN to PNx, DNa, requests the block from DNb. Since the PNs connected to DNa will possibly request the same block from DNa, it is most important that this block be kept as long as possible in the DNa's buffer to prevent unnecessary disk accesses and DNb-DNa transfer(s). The priority scheme sought is such that a block from the most distant DN has the highest priority. However, the blocks of a DN just read from its disk should not be discarded, as they will be shared by the PNs connected to this DN. Different buffer management mechanisms have been currently investigated.

The design of the I/O organization and the software interface must clearly be determined in relation to applications, for it is the flow of data files at the request of the application's needs that must determine the appropriate number of processors for each disk and also the file flow among the interconnected disks. The file system must be such that for the PNs the concept "file" must be meaningful.

The interconnection of DNs affects the scalability of the system and file usage for different applications. Intuition indicates another hypercube of smaller dimension than the dimension of the PN hypercube. However, our research will try other networks, such as mesh and ring. Which DN should be connected to which PN and which blocks should be stored on which DN depend on the application running. They depend on the locality of PN programs. The file domain of each PN must be mirrored closely by them. File blocks needed by a cluster of PNs should be on the closest DN so that file access time is not long.

A brief explanation of the matrix multiplication algorithm developed for a hypercube with a file system is to be given. Two N x N matrices A and B are multiplied and the result is contained by C. Each matrix is a file and each row is a single file block. The physical system is the same as the one given in figure 1. A hypercube connection is assumed among the DNs. The algorithm has the following steps:

1) The IH distributes A and B matrices to the DNs. The B matrix is completely stored on each DN as shown in figure 2. This is not possible with matrix multiplication algorithms written for conventional hypercubes. The way the C file is expected on the DNs is also shown in figure 2.

2) Each DN distributes an equal amount of the A matrix to its PNs as shown in figure 3.

3) Each PN calculates the assigned rows of matrix C by multiplying their A rows with all the rows of matrix B. Each PN reads row I of B from its DN and multiplies the row with the I^{th} element of the A matrix. When a PN finishes, it returns its results to its DN.

4) When a DN receives results from its PNs, it sends the results to the IH.

The above algorithm does not set limit on the size of the matrices. If the size is too large for the hypercube of PNs, the rest of the matrices is read from the IH while the PNs are involved in computations in step 3. The efficiency of the system is increased due to the overlapping or macro pipelining [11] and can be increased more by more overlapping such as steps 2, 3 and 4.

Also note the sharing of file blocks in the third step of the algorithm. The PNs of each subcube share the same blocks and need them at approximately the same time. The DN which responds to the first block request from a PN by reading from its disk, will respond to the requests for the same block from its other PNs without reading from its disk but by reading from its buffer.

The development of this algorithm has shown a new fact about writing algorithms for a hypercube with a file system. The decomposition process which is influenced by load balancing, concurrency and communication is also influenced by the nature of file blocks. The process now has to consider efficient use of blocks distributed over the DNs. A block, a sequential set of data elements, must be used in enough number of computations to justify its movement. This is very dependent on how the blocks partition the data domain. (Should a block be a row, a column or a submatrix ?)

A hypercube simulator has been obtained from ORNL [12]. It simulates a hypercube similar to the Intel iPSC hypercube. It helps programmers develop and debug hypercube programs which must be

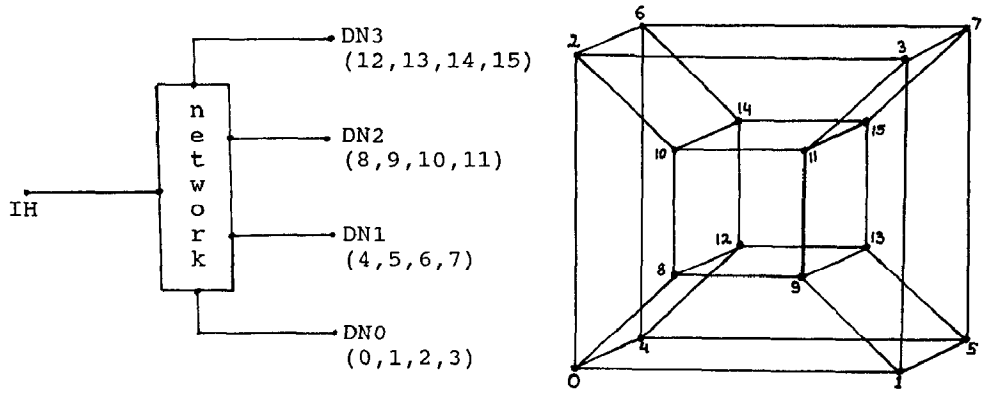

Figure 1: The I/O organization of the hypercube file system

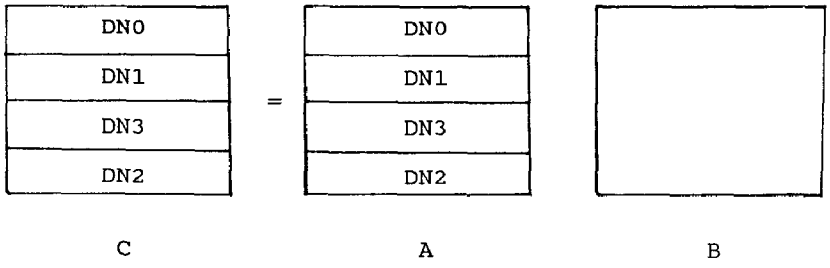

Figure 2: The distribution of all the matrices on the DNs

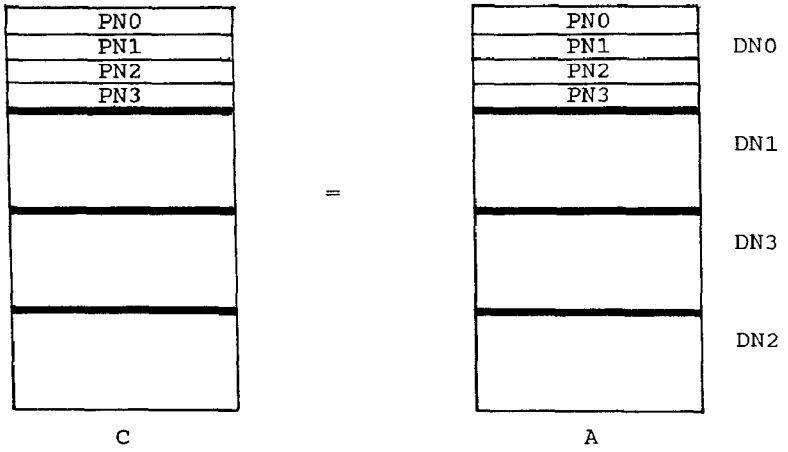

Figure 3: The distribution of the A and C matrices on the PNs

written in FORTRAN or C. It has a post-processor which produces performance results about the execution of application programs. The simulator has some machine language code that forces the user to run the simulator on a VAX/11-780 machine or alike.

The ORNL simulator cannot simulate a hypercube with a file system. It has been modified so that at the request of the user, it can simulate a hypercube with a file system [13]. Currently the simulator is used for developing algorithms and for obtaining performance results for both types of hypercubes.

5 Conclusion

A distributed hypercube file system is proposed for a better space/time tradeoff. The I/O organization of the system is that a number of disk nodes are interconnected and each disk node is connected to a subcube of processing nodes. Files are distributed across the disk nodes and large data transfers are accomplished in file blocks between a disk node and a processing node without using any interprocessor link. Message transfers are supplemented by file block transfers for data transfers. The interconnection of disk nodes enables any processing node to access any file block fast. The software interface of the file system manages disk node buffers and directories. Coherence, synchronization and locking problems are resolved by the software interface.

The real advantage of the presented file system is that it expands the class of problems that can be applied to a scalable hypercube system. It is apparent that such real-time problems as image processing require immense amount of I/O bandwidth and the presented hypercube can support such applications. The value of a hypercube with a file system will be apparent more when complete applications are run. A complete application has a number of stages. Each stage may have a different decomposition. A hypercube system must be able to switch from one decomposition to another decomposition fast. This process will be easier in the system proposed in this paper. Further advantages of the file system, if projected, can be in implementing virtual memory for processing nodes. Multiprogramming and multiprocessing of subcubes and individual processing nodes and in implementing such communication types as global send, scatter and gather [14]. Dynamic load balancing is possible through DN-DN links. Each disk node does not have to be identical. Instead, some may be specialized in certain applications as mentioned in [4]. A separate host can be connected to each disk node. Therefore, these new disk nodes can be renamed as Extension Nodes (ENs).

Work is underway to determine the best way of interconnecting the disk nodes, connecting a disk node to a subcube of processing nodes, managing disk node buffers and directories. Besides the matrix multiplication algorithm three other problems are chosen to study the effect of a distributed file system. These problems are matrix inversion, Cholesky factorization and the Fast Fourier Transform. Algorithms for these problems are being developed for a hypercube with and without a distributed file system. New metrics and formulations for the distributed hypercube file system will be developed to predict the performance of the system.

References

[1] Seitz, Charles L., "The Cosmic Cube", Communications of the ACM, January 1985, pp. 22-33.

[2] Lang, Charles R., "The Extension of Object Oriented Languages to a Homogeneous, Concurrent Architecture", Ph. D. thesis, Technical Report Number 5014, California Institute of Technology, Pasadena, May 1982.

[3] Fox, G. Fox and Frey, Alexander, "Features of a TeraFLOPS Supercomputer", ACM Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, to be published, 1988.

[4] Witkowski, A. and et al, "Concurrent I/O System for the Hypercube Multiprocessor", ACM Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, to be published, 1988.

[5] NCUBE Users Handbook, Beaverton, NCUBE, October 1987.

[6] Close, P. and Wolper, A., "The iPSC 80386-Based Hypercube Node", ACM Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, to be published, 1988.

[7] Fazzari, Rodney J. and Lynch, John D., "The FPS Mark II T Series: An Enhanced Parallel Vector Supercomputer", ACM Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, to be published, 1988.

[8] Fox, Geoffrey C. and Otto, Steve W., "Algorithms for Concurrent Processors", Physics Today, May 1984, pp. 50-59.

[9] Nugent, Steven F., "The iPSC/2™ Direct-Connect™ Communications Technology", ACM Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications, to be published, 1988.

[10] Fox, Geoffrey C. and Otto, Steve W., "Matrix Algorithms on the Hypercube", Research Report Number 297.3, Caltech Concurrent Computation Project, California Institute of Technology, Pasadena, November 1985.

[11] Hwang, Kai and Briggs, F. A., "Computer Architecture and Parallel Processing", McGraw-Hill, 1984.

[12] Dunigan, T. H., "A Message-Passing Multiprocessor Simulator", Oak Ridge National Laboratory, Oak Ridge, May 1986.

[13] Mowatt, P., "Development of a Hypercube File System", Summer Research Project Report, Polytechnic University, New York, September 1987.

[14] Saad, Yousef and Schultz, Martin H., "Data Communication in Hypercubes", Research Report YALEU/DCS/RR-428, Yale University, New Haven, October 1985.