LA-UR -86-1151

CONF-8508173--1

~~RECEIVED BY OSTI~~ MAY 12 1986

LA-UR--86-1151

DE86 010164

TITLE: # A HYPERCUBE PROJECT AND A SIMULATOR FOR A HYPERCUBE OF COMPUTERS

AUTHOR(S) Jung Pyo Hong, Robert D. Tomlinson, Nisheeth Patel and L. Howard Pollard

SUBMITTED TO Proceedings of the Conference on Hypercube Microprocessors held in Knoxville, TN (8/85) in SIAM Publications.

[illegible]

A HYPERCUBE PROJECT AND A SIMULATOR FOR A HYPERCUBE OF COMPUTERS

Jung Pyo Hong* and Robert D. Tomlinson,* Nisheeth Patel,† and L. Howard Pollard‡

Abstract. The Los Alamos National Laboratory and the University of New Mexico are working on HC, a hypercube project for large problems. A hypercube of 1024 nodes is being designed; implementation of a machine of at least 16 nodes is underway. The nodes of the machine are complete computer systems, including mass storage and a large main memory.

The computational abilities of each node are enhanced by using a detached instruction fetch-execute-store organization. This allows the execution rate to achieve a consistently high level, approaching 90% of optimum. Trade-offs in the implementation methods indicate the speeds available with acceptable hardware investments.

A simulator has been written which allows execution of high level programs on a hypercube organization. The simulator has shown that programs prepared for execution on a single stream machine can be easily modified to use the resources of a parallel machine, and that the results are identical to previous executions.

Introduction. A hypercube[1] of computers was discussed in the mid 1970's, and the Russians[2] built a 32-node hypercube in the late 1970's. In addition to these instances, there are other references to hypercubes[3,4] in the literature. Researchers at Caltech built a 64-node hypercube in the early 1980's. Their efforts, along with others, have demonstrated that many problems in physics[5,6,7,8,9,10,11,12,13,14,15,16,17,18,19] can be implemented efficiently on a hypercube.

---

* Los Alamos National Laboratory, Electronics Division, E-10 Data Systems

† U.S. Army Ballistic Research Laboratory

‡ University of New Mexico, Department of Electrical and Computer Engineering

This work is supported by the U.S. Army Ballistic Research Laboratory under Project Orders BRL PO 54-86, BRL PO 76-86 and BRL PO 18 86, and by the U.S. Army Research Institute for the Behavioral and Social Sciences under MIPR 13 ARI 86-52.

The hypercube is a structure in which there is a node at each vertex of a cube in n–th order space. When referring to a computer organization, each node in this hyperspace contains a computational element. In the HC project at Los Alamos National Laboratory and the University of New Mexico, we are using this organization to provide sufficient computing resources to address some complex problems. One approach is to implement the parallelism by utilizing a simple CPU as the computational element. Instead of using nodes with minimal computing resources, we have elected to implement the architecture using nodes with sufficient computing resources to address interesting problems. Thus, each node is itself a complete computer system, with a disk and an operating system.

This ensemble of computers is then connected to form a hypercube in R space by connecting each of the $N = 2^R$ computers to its R logical nearest neighbors. Communication is accomplished by asynchronous writes and synchronous reads. That is, a node generates a message whenever it needs to send information to another node, and then continues with its computation. However, when information is needed by a node to continue its work, then the node waits until that data is available before proceeding. This read mechanism is the only method available for bringing the nodes in step with each other.

Hardware Description. One of the attractive features of an ensemble architecture such as the hypercube is that a machine can be sized to fit problems of a particular nature. At Los Alamos National Laboratory we have been looking at a class of problems for the Balistics Research Laboratory of the Army and others which requires a large amount of computing resources, and which maps well onto the structure of the hypercube. An order 10 hypercube (1024 nodes) is being designed to address these problems. This machine will have sufficient computational capability to address real problems of a size which would not fit well on other architectures. An order 4 machine (16 nodes) will be built to provide a test vehicle for experimenting with the concepts involved; a full machine will be implemented only when the preliminary portions show positive results.

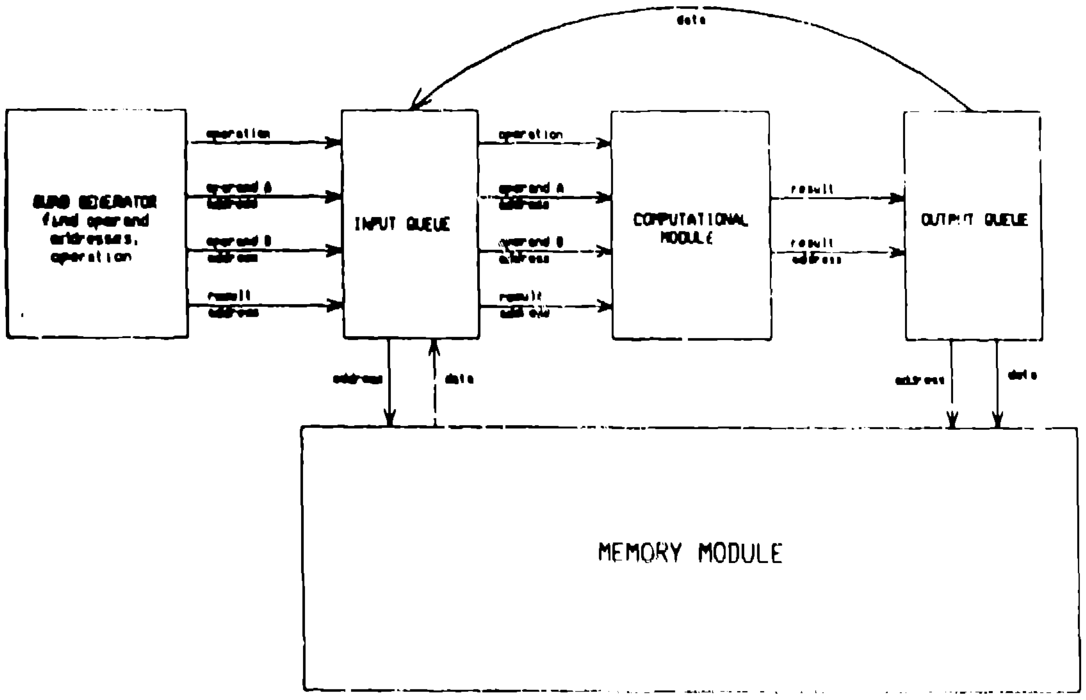
Rather than approach the problem of implementing parallelism by using nodes with minimal computing resources, we have elected to implement the architecture using nodes with sufficient computing resources to address interesting problems. Thus, each node is itself a complete computer system, capable of running UNIX;* the actual system software will start with UNIX and be cut down to remove unneeded overhead. All of the nodes in the machine will be identical, with the exception of a number which will uniquely identify the machine within the structure and also identify its nearest neighbors, connected to it by direct links. Not only are all nodes identical, but there is no "host" machine; each node has the capability to perform the work done by a host.

Each node is composed of two sections: a section which is responsible for all of the I/O and communications, and a computational section which has been optimized for scientific problems. In addition to a control CPU, the I/O section contains

* UNIX is a trademark of AT&T

a disk controller, an Ethernet port, links to other nodes, and a port into the memory of the node. Each node will have attached to it a large disk, with a capacity of greater than 300 million bytes, capable of transfer rates of 2.4 million bytes/sec. This disk can be used to store data and results as needed, as well as whatever system programs are necessary. The Ethernet access is intended for development and engineering access, and not for use as a general resource when the machine is operating as a hypercube; however, it can be used to transfer data to and from the nodes and disks when preparing for a computation or examining results. The links are parallel ports which can transfer data between nodes at rates in excess of 5 million bytes/sec; however, the aggregate data rate of all of the links at a node is limited by the bandwidth of the bus in the I/O section. Currently this limit is 10 million bytes/sec, but the limit could be increased if studies indicate that a higher inter-node communication rate is needed.

The computational section has been designed to utilize currently available technology to speed up overall data rates. A conceptual block diagram of the section is shown in Figure 1.



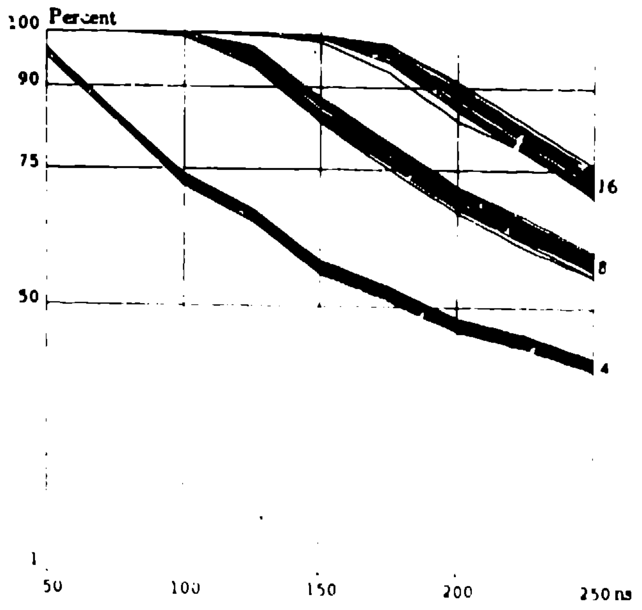
Conceptual Block Diagram of Hypercube Node
Los Alamos National Laboratory
FIG. 1.

This organization is similar to the technique proposed by Smith[20]. The Quad Generator is responsible for directing the sequence and action of the computations. Each computation is identified by an operation to be performed, the operands on which to perform this work, and the location of the destination. This information is given to the Input Queue, which is responsible for obtaining the operands needed in the computation, either from the Memory Module or from results of a previous operation which is available in the Output Queue. The Memory Module contains between 16 and 64 million bytes of memory, sufficient to hold the data needed for large and interesting problems. This memory is organized in multiple banks to permit efficient access for information transfers. As the operands become available they are supplied, along with the operation to be performed, to the Computational Module. The Computational Module contains two AMD29325 floating point computational units which are connected to permit calculation of $\sum X_i Y_i$ at the rate of one term per cycle. Assuming a clock rate of ten megahertz, this is a computational rate of twenty megaflops. When results are available, they are supplied, along with the destination information, to the Output Queue module, which is responsible for directing the results to the proper location. All of these modules work together under the direction of a 32032 house keeping computer, which is responsible for initiating the work which needs to be done.

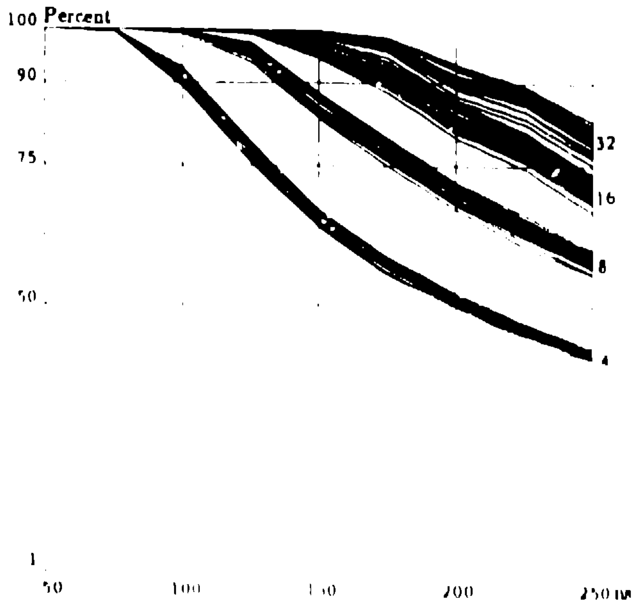
The effect of detaching the instruction fetch and decode from operand access and instruction execution is to speed up the effective computational rate of the system. The Quad Generator performs much of the work which would normally be done by the instruction execution unit in a conventional architecture. This leaves the Computational Module free to perform the computations needed to solve the problem. The Quad Generator can cause the operations to be performed in an order similar to a vector type of operation, or the operations could be performed in a sequence more closely matched to the underlying problem being solved.

The size and capability of the various portions of the system have been under study, and matching functional capability with buildable technology remains a topic of interest. If the various components are kept small they can be built with few problems; however, the performance may not approach the desired level. On the other hand, the implementation problems associated with a more complex unit may render the whole machine unbuildable or unreliable. Studies of the trade-offs involved are underway, and some of the results are shown in Figures 2 and 3. Figure 2 gives an indication of the effect that the number of banks in the memory has on system performance. The graph shows the percentage of full load for the fast floating point unit for different memory speeds. The lines which make up each band represent different probabilities that a calculation will use an operand identified in a calculation already in the pipe. Each band represents a division of the memory into a different number of banks. As expected, increasing the banking factor does indeed improve the system throughput. However, the graph also gives some insight into the tradeoffs between faster and more expensive memory, and slower memory with more banks. Figure 3 is a look at another facet: the effect of the length of the input queue on the rate of calculation. Increasing the length of the input queue does indeed improve performance, but the graph indicates the type of performance improvements can be expected by doubling the hardware investment. More and closer looks at the various design parameters will be needed before

the machine can be further defined.



ALU Performance as a Function of Number of Banks
(Input Queue Length = 8)
FIG. 2.



ALU Performance as a Function of Input Queue Length
(Input Queue Length 8)
FIG. 3.

Hypercube Simulator. In order to explore the space of communication and computation problems associated with putting a problem on the hypercube, a simulator capable of large, and hence realistic, program execution has been developed. This simulator has been used to show that certain important classes of numerical computations match closely the hypercube organization. Hydrodynamics and Monte Carlo programs have been executed using the simulation system.

The simulator executes in the 4.2 BSD UNIX environment, where each node of the hypercube is a process. Within the 4.2 BSD UNIX environment, this guarantees that there will not be accidentally shared memory, resulting in an exact model of how the hypercube will be physically arranged. The user process can create, read, and write files. In face, ordinary programs can be executed on a node. Within a node, all UNIX programming tools and conventions are available, including forking. There need not be any new language nor operating system for the node. Very large programs may be written for each node.

Inter-node communications are simulated by sockets. From the user's point of view, a node does a read or write to another node. The use of the socket formalism means that programs written for the simulator will execute on the hypercube hardware.

The simulator has executed programs written in Fortran and C. Once again it is stressed that no special programming language is required. One example of a program that was executed on the simulator is about one thousand lines of ordinary Fortran composed of eight subroutines. It is computationally intense, there being a sequence of over one hundred fifty thousand floating point operations in one subroutine.

A Fortran program, written before the author of the code was aware of the hypercube simulator, was put on the hypercube simulator with small modifications. The output from a simulation on 16 nodes was exactly the same as the original output from the linear program, bit for bit. It showed that the number of changes that had to be made was small. It also showed that the inter-node communications and the synchronizations required for this code.

Conclusion. We are in the process of designing a large scale hypercube of computers, the operation of which will be an effective tool in the study of large scale problems. The nodes of this hypercube will be complete computer systems, giving the overall machine enormous capabilities for storage and computational complexity. Our studies have shown that the disk at every node will be effectively utilized, and a system resource which will enhance the capabilities of the machine.

The machines at each node will contain high speed arithmetic units to provide enhanced throughput capabilities. The detaching of the instruction decoding from the operand access and instruction execution provides a means of streaming data through the high speed arithmetic units. The optimum organization of the constituent parts into a system is still under investigation, but our studies indicate that we should be able to keep the arithmetic units busy 90% of the time.

We have written a simulator which allows execution of real, non-trivial programs to study the decomposition of problems onto the hypercube structure. This simulator has shown that programs with inherent parallelism can be prepared for the hypercube organization with few changes and compatible results.

REFERENCES

[1] W. Millard, hyperdimensional micro-p collection seen functioning as mainframe, Digital Design, vol. 5, no. 11, p. 20, 1975.

[2] N. M. Allakhverdiyev and S. S. Sarafaliyeva, Choice of Multiprocessor System Configuration for Digital Signal Processing, SR Report - Cybernetics, Computers and Automation Technology, Foreign Broadcast Information Service, July 7, 1983.

[3] David J. Evans, in Parallel Processing Systems, pp. 227-228, Cambridge University Press, 1982.

[4] R. W. Hockney and C. R. Jesshope, in Parallel Computers, pp. 10, 42, 321, 323-324, Adam Hilger, Ltd., 1981.

[5] Geoffrey C. Fox and Steve W. Otto, algorithms for concurrent processors, Physics Today, May 1984.

[6] Geoffrey Fox, Annual Report of the Caltech Concurrent Processor Project July 1984, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[7] Greg Lyzenga, The Nearest Neighbor Concurrent Processor: A User's Tutorial Guide August, 1984, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 10, 1984.

[8] B. T. Werner and P. K. Haff, Grain Dynamics Simulations on the Caltech Concurrent Processors, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[9] Steve W. Otto, Lattice Gauge Theories on a Hypercube Computer, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[10] Edward Felton, Scott Kariin and Steve Otto, Sorting on the NNCP, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[11] F. Fucito and S. Solomon, On the Relation Between the Coulomb Gas and the Lattice XY Model April 3, 1984, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[12] F. Fucito and S. Solomon, Long Range Forces on NNCP, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[13] F. Fucito and S. Solomon, Monte Carlo Parallel Algorithm for Long Range Interactions June 7, 1984, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30,

1984.

[14] Mark Alan Johnson, A Statistical Physics Simulation on the Hypercube, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[15] John Salmon, An Astrophysical N-body Simulation on the Hypercube, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[16] D. Meier, Two-Dimensional, One-Fluid Hydrodynamics: An Astrophysical Test Problem for the Nearest Neighbor Concurrent Processor July 22, 1984, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[17] Robert W. Clayton, Finite Difference Solutions of the Acoustic Wave Equation on a Concurrent Processor, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[18] Paul Hipes and Aron Kuppermann, Simulation of Atom-Diatom Collisions on Parallel Computers, in Caltech/JPL Concurrent Computation Project Annual Report 1983-1984 and Recent Documentation, August 30, 1984.

[19] Jung P. Hong, Robert D. Tomlinson and Nisheeth Patel, An Example of How to Use the Hypercube Simulator with a Fortran Program, in Formal Report LA-UR-1405, Los Alamos National Laboratory, April 18, 1984.

[20] James E. Smith, decoupled access/execute computer architectures, ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984.