Architecture of a Hypercube Supercomputer

John P. Hayes, Trevor N. Mudge, Quentin F. Stout

Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Dept.
University of Michigan
Ann Arbor, Michigan 48109

Stephen Colley, John Palmer

NCUBE
1815 N. W. 169th Place, Suite 2030
Beaverton, Oregon 97006

Abstract: The implementation of massively parallel computers based on hypercube architectures is discussed in this paper. It is argued that such machines offer an alternative to traditional supercomputers at far lower cost. The rationale for using hypercube machines for supercomputing applications is examined, including cost, node performance, communication speed, packaging, reliability, and programming requirements. These issues are illustrated by a recently introduced commercial hypercube supercomputer, the NCUBE/ten. The major design decisions underlying the NCUBE/ten's implementation technology, system architecture and operating system are described.

1. Introduction

A hypercube or (binary) n-cube computer is a multiprocessor characterized by the presence of $N = 2^n$ processors interconnected as an n-dimensional binary cube. Each processor P_i forms a node (vertex) of the cube and is a self-contained computer with its own CPU and local main memory. P_i has direct communication paths to n other processors (its neighbors), which correspond to the edges of the cube that are connected directly to P_i. 2^n distinct n-bit binary addresses or labels may be assigned to the processors so that each processor's address differs from that of each of its n neighbors in exactly one bit position. Figure 1 illustrates the hypercube topology for $n < 4$; note that a zero-dimensional hypercube is a conventional SISD computer. An n-dimensional hypercube Q_n for $n \geq 2$ can be defined recursively in terms of the graph product operation $\times$ as follows [4], where $K_2 = Q_1$ is the complete 2-node graph:

$$Q_n = K_2 \times Q_{n-1}$$

As illustrated by Fig. 1, Q_n is composed of two copies of Q_{n-1}. Every node P_{0x} in one copy of Q_{n-1} is the neighbor of a node P_{1x} in the other copy.

For some time, it has been known that the hypercube structure has a number of features which make it a useful architecture for parallel computation. For example, meshes of all dimensions and trees can be embedded into a hypercube so that neighboring nodes are mapped to neighbors in the hypercube. The communication structures used in the Fast Fourier Transform and Bitonic Sort algorithm can similarly be embedded into the hypercube. Since a great many scientific applications use mesh, tree, FFT, or sorting interconnection structures, the hypercube is a good candidate for a general-purpose parallel architecture. Even for problems with less regular communication patterns, the fact that the hypercube has a maximal internode distance (the graph diameter) of $n = \log_2 N$ means any two nodes can communicate fairly rapidly. This diameter is larger than the unit diameter of a complete graph K_N, but is achieved with nodes having only degree or fanout of $\log_2 N$, as opposed to the $N - 1$ degree of nodes in K_N. Other standard architectures with small degree, such as meshes, trees, or bus systems, either have a large diameter ($N^{1/2}$ for a 2-dimensional mesh) or a resource which becomes a bottleneck in many applications because too much communication must pass through it (as occurs at the apex of a tree, or at a large shared bus). Thus, from general topological arguments it can be concluded that hypercube architectures offer a good balance between node connectivity, communication diameter, algorithm embeddability, and programming ease. This balance makes them suitable for an unusually wide class of computational problems.

Based on various considerations of the foregoing kind, proposals to build large hypercube computers have been made for more than twenty years. In 1962, Squire and Palais at the University of Michigan, motivated by the hypercube's rich interconnection geometry and programming ease, carried out a detailed paper design of a hypercube computer [13,14]. They estimated that a 4096-node (12-dimensional) version of their machine would require about 20 times as many components as the IBM Stretch, one of the largest and most complex contemporary computers. Around 1975 IMS Associates, an early manufacturer of personal computers, announced a 256-node commercial hypercube based on the Intel 8080 microprocessor, but its design details were not published and the machine was never produced. In 1977, Sullivan et al. presented a thorough analysis of hypercube architectures, and a proposal to build a large hypercube

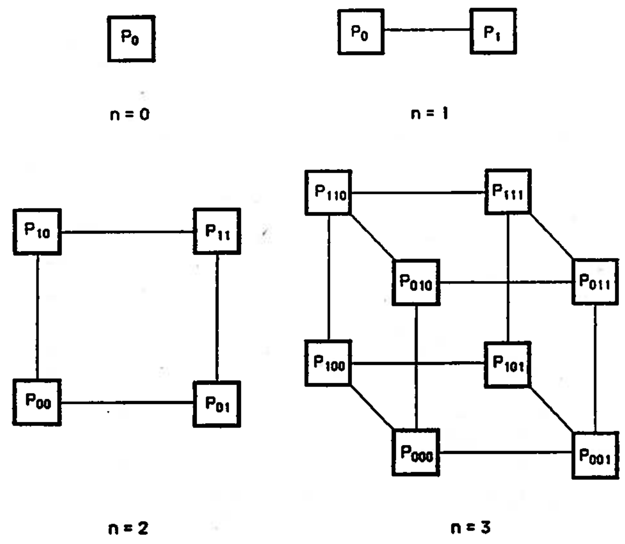

Fig. 1. n-dimensional hypercube for n=0,1,2,3.

Presented at the 1986 International Conference
on Parallel Processing

called CHOPP (Columbia Homogeneous Parallel Processor) containing up to a million processors [15,16]. In the same year, Pease published a study of the "indirect" binary n-cube architecture, in which a multistage interconnection network of the omega type is suggested for implementing the hypercube topology [8]. A number of other interesting architectures closely related to the hypercube have also been proposed, for example, the cube-connected-cycles structure [11].

It is clear that the early hypercube designs were impractical because of the the large number of components (logic and memory elements) they required using the then available circuit technologies. The situation began to change rapidly in the early 1980's as advances in VLSI technology allowed powerful 16/32-bit microprocessors to be implemented on a single IC chip, and RAM densities moved into the 10^5–10^6 bits/chip range. A working hypercube computer was not demonstrated until the completion in 1983 of the first 64-node Cosmic Cube at Caltech [12]. For the hypercube node processor, it uses a single-board microcomputer containing the Intel 8086 16-bit microprocessor and the 8087 floating-point co-processor. Since then, Caltech researchers have built several similar hypercubes, and successfully applied them to numerous scientific applications, often obtaining impressive performance improvements over SISD machines of comparable cost [3].

Influenced primarily by the Caltech work, a number of commercial hypercubes have been developed since 1983. In July 1985, Intel delivered the first production hypercube, the 128-node iPSC (Intel Personal Supercomputer) which has a 16-bit 80286/287 CPU as its node processor. Assuming a peak performance of 0.07 MFLOPS per node, the 128-node iPSC has a potential throughput of about 8 MFLOPS, far below that of a traditional vector supercomputer such as the Cray-1 (160 MFLOPS). Other commercial hypercubes were also introduced in 1985 by Ametek Inc. and NCUBE Corp. The Ametek System/14 hypercube can have up to 256 nodes, which employ an 80286/287-based CPU similar to that of the iPSC, with the addition of an 80186 processor for communication management. The NCUBE/ten can accommodate up to 1024 nodes, each based on a VAX-like 32-bit custom processor with a peak performance of 0.5 MFLOPS. Thus a fully configured NCUBE system has a throughput potential of around 500 MFLOPS. This high performance level is supported by extremely fast communication rates (both input/output and node-to-node) making the NCUBE/ten a true supercomputer. NCUBE machines have been installed at several beta test sites, including the University of Michigan, since early 1985, and have been in general production since December 1985. Several other hypercube-style machines with supercomputing potential are presently under development, including the Caltech/JPL Mark III [9] and the Connection Machine [5]. Much faster successors to the current commercial hypercubes can also be expected to appear over the next few years. Because of the effort being devoted to the development of hardware and software for these machines, and their relatively low cost, hypercube supercomputers seem likely to provide an increasingly attractive alternative to conventional pipelined supercomputers for many applications.

This paper explores the architectural and technological issues influencing the design of supercomputing hypercubes, with the NCUBE/ten serving as an example. Particular attention is devoted to the influence of component packaging, reliability, communication speed, and the operating system environment on the system implementation. Section 2 discusses the general design requirements of hypercube supercomputers, while the specific design decisions made for the NCUBE/ten are covered in Sec. 3. Software issues are discussed in Sec. 4.

2. General Design Issues

Supercomputing performance requires extremely high integer and floating-point execution rates, as well as extremely high I/O throughput. Very large primary (RAM) and secondary (disk) memory spaces are also usually required. For the principal user base of scientific programmers, the programming environment needs to provide FORTRAN, and a powerful operating system such as UNIX. Low cost and high reliability imply minimizing the component count at all levels, particularly the numbers of chips and boards used. Some degree of fault tolerance is also very desirable. Since a very large amount of RAM storage is needed, memory fault detection and correction via an error-correcting code (ECC) is an important consideration, despite the fact that it increases the chip count. Reliability is increased, and operating cost decreased, by employing an air-cooled configuration suitable for a standard office environment. Based on an examination of various existing computer systems, it can be concluded that the air cooling limits the machine complexity to under 50,000 chips. Off-the-shelf parts decrease costs and usually increase reliability. If custom chips are needed, then computer manufacturers who rely on outside suppliers should use conservative design rules which will be accepted by multiple silicon foundries. In large-scale numerical calculations, the possibility of large error accumulation forces the individual calculations to be as accurate as possible. Numerical accuracy can be increased by adhering to the IEEE 754 floating-point standard and providing double-precision floating-point operations.

A key decision in the design of a parallel computer is the choice of the interconnection network to be used. Multistage interconnection networks have been advocated as simplifying the programming process by providing a global shared memory, but it did not seem possible to build a large multistage network using the technology available in 1983 without suffering significant delay in passing information through the network. Since this strongly affects performance, it was felt that to achieve supercomputer performance it would be necessary to use a direct connection network with local memory at every node. Many direct interconnection schemes have been analyzed and implemented but, as discussed in Sec. 1, the hypercube structure has a number of inherent advantages. The ease with which efficient application programs were developed for the hypercubes at Caltech has also shown the hypercube to be superior to alternative architectures such as meshes or trees. The neighbor-to-neighbor links of the hypercube provide almost the same communication capabilities as a complete graph, while using nodes with only a logarithmic degree. The achievable degree is constrained by a variety of packaging considerations, but with current technology it is possible to build hypercubes with thousands of nodes. In contrast, a complete graph connection of a few tens of nodes may not be possible.

There are additional features of the hypercube that are particularly useful in designing a supercomputer, but have not been previously exploited. For example, the hypercube is homogeneous in that all nodes look the same, hence it is natural to attach an I/O channel to each node. This provides the potential of extremely high system I/O rates. Also, since there are numerous ways to divide a hypercube into subcubes, it is easy to support multiprocessing where each user has a dedicated subcube. These subcubes can be allocated so that all processor-to-processor and I/O communications occur without using processors or communications lines in other subcubes. Further, by writing programs in which the size of the subcube is a user-defined parameter, it is possible to develop programs in small subcubes and then do production runs in larger subcubes. This partitionability also makes it easier to tolerate faults, since the operating system can allocate subcubes which avoid faulty processors or faulty communication lines.

As discussed in Sec. 1, technology developments have enabled a hypercube computer to be built reliably with a large number of processors. A fine-grained supercomputer architecture, i.e., one with a large number (over 1000) of very simple processors, has a high ratio of communication to computation. The Connection Machine is an example of a fine-grained hypercube-class computer, but its suitability for scientific computations is unclear. A very coarse-grained architecture with, say, 10 to 100 large and fast processors, requires that the nodes achieve extremely high performance. For example, to achieve 10^9 instructions/sec with 10 processors requires processors capable of 10^8 instructions/sec. The Caltech/JPL Mark III will be an example of a coarse-grained hypercube. It was felt by the designers of this machine that achieving 10^9 instructions/sec is best done with 1000 processors running at 10^6 instructions/sec.

Experience with the Caltech machines has demonstrated that a medium-grain MIMD hypercube architecture can attain high efficiency on a variety of scientific problems, with a tolerable amount of revision of serial code and algorithms [3]. This can be contrasted with the much greater amount of program and algorithm redesign required of users of fine-grained SIMD machines such as the MPP [10]. MIMD machines require each node processor to perform instruction fetching, decoding, and other functions that are not performed by SIMD nodes. Distributed-memory MIMD machines must supply a program to each node, so that MIMD machines may pay a large penalty in chip area and chip count. In general, for the same chip area and number of chips one can build more SIMD processors and have a greater potential system throughput; however, the gain in programming simplicity obtained by using an MIMD machine more than compensates for this, except for a narrow range of applications in which almost any penalty can be tolerated if it yields the required speed. Furthermore, MIMD machines can accommodate multiple independent users, while SIMD machines cannot.

Since there may be hundreds or thousands of nodes in a hypercube supercomputer, their chip count is the most significant component of the total system chip count. Using the densest possible memory chips available is the key factor in decreasing the number of chips. The NCUBE/ten, for example, uses 256K DRAM chips to implement the local memories of the hypercube nodes. The next most significant reduction in chip count can be achieved by putting all node functions onto a single chip. This implies that the processor chip must perform all communication, memory management, floating-point operations, and other data-processing functions. Unlike RAMs, there is not yet widespread market pressure to produce standard processor chips of this type, consequently, they are not available off the shelf. In 1983, when design of the NCUBE/ten started, the only way to achieve supercomputer performance with a one-chip node processor was to undertake the risky step of custom-designing such a complex chip. INMOS made a similar decision with the Transputer processor chip, with the important difference that the initial version of the Transputer does not provide floating-point operations, and has four rather than eleven I/O channels [6]. The performance and functionality demands on the NCUBE processor chip are quite severe, and numerous tradeoffs were needed to enable it to be built with current technology.

3. NCUBE/ten Architecture

The overall goal of the NCUBE designers was to use massive parallelism to build an inexpensive and reliable range of software-compatible machines achieving supercomputer performance at the high end. The largest model in the series, the NCUBE/ten, is a 10-dimensional hypercube containing 1024 powerful 32-bit processors of custom design, each with a 128-Kbyte local memory. Up to eight front-end host processors are used to manage I/O operations under control of a multiuser UNIX-based operating system. An unusually high level of system integration is employed that allows 64 processors with their memories and interconnections to be placed on a single printed-circuit board. A maximum-sized NCUBE/ten system is composed of 16 processor and 8 I/O boards (including host processors) and is housed in a small air-cooled enclosure.

The NCUBE node processor provides the functions of a 32-bit supermini-class CPU, including a full floating-point instruction set, and all the logic needed for memory management and interprocessor communication on a single VLSI chip; see Fig. 2. The design of the processor was started by NCUBE in 1983, constrained by the desire to use conservative design rules acceptable to several silicon foundries. The chip was designed using 2 μm (approx.) nMOS design rules, and contains about 160,000 transistors. It is housed in a pin-grid-array package with 68 pins. Combined with six 256K-bit DRAM chips (each of which is organized as 64K x 4 bits), an entire NCUBE/ten node requires only seven chips. Because of this, a 6-dimensional hypercube with 64 nodes and 8 Mbytes of memory can be packed into a single 16" $\times$ 22" board, a photograph of which appears in Fig. 3. The backplane connections are rather formidable since each node has off-board bidirectional channels to four more processors of the hypercube, plus one bidirectional channel to an I/O board, resulting in 640 backplane connections just for communication channels.

The instruction set of the NCUBE/ten is conventional and quite orthogonal, being similar to the VAX instruction set without the latter's 3-address addressing modes [7]. There are three main classes of information: addresses (unsigned integers), integers and floating-point numbers (reals). Addresses are 32 bits long, but the current node implementation only supports a 17-bit physical address space. Integers can be 8, 16 or 32 bits long. Floating-point numbers can contain either 32 or 64 bits, and conform to the IEEE 754 floating-point standard. There are 16 general-purpose registers of 32 bits each. A variety of addressing modes are available, including literal (immediate), register

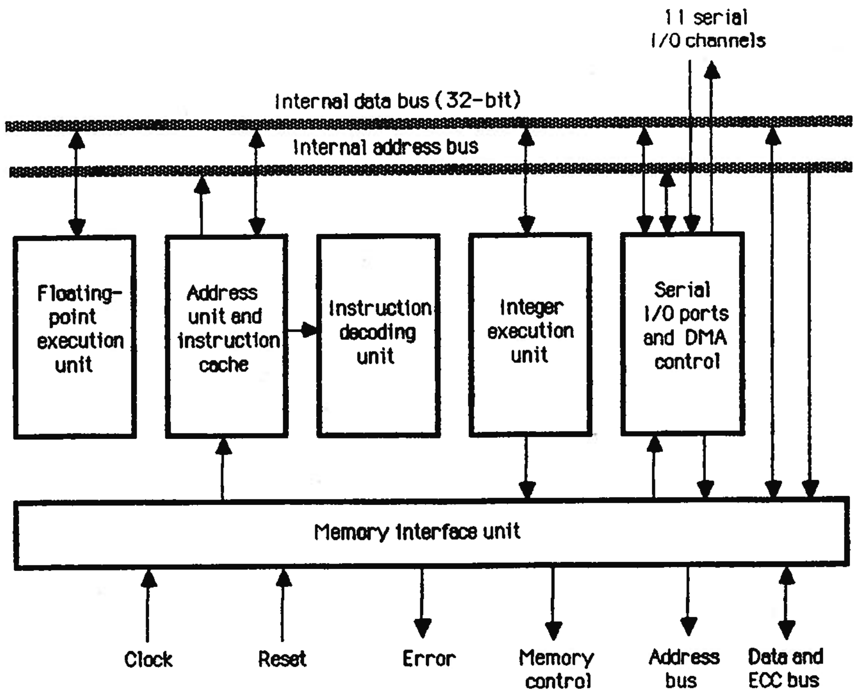
11 serial I/O channels
Internal data bus (32-bit)
Internal address bus
Floating-point execution unit
Address unit and instruction cache
Instruction decoding unit
Integer execution unit
Serial I/O ports and DMA control
Memory interface unit
Clock
Reset
Error
Memory control
Address bus
Data and ECC bus

Fig. 2. Organization of the NCUBE processor chip.

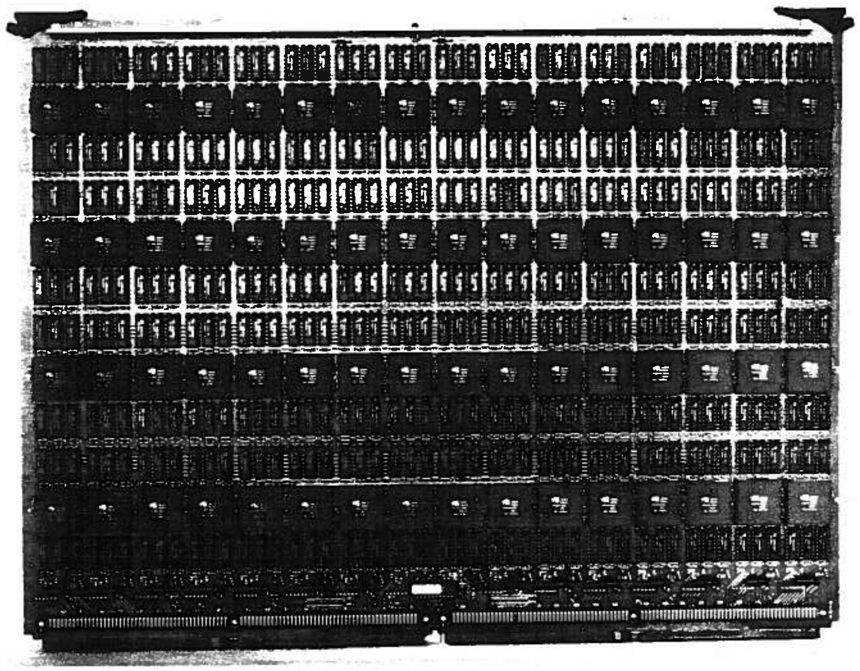

Fig. 3. The 64-node NCUBE processor board.

direct, autodecrement/increment, autostride, offset, direct, indirect, and push/pop. The instruction set contains a full complement of logical, shift, jump, and arithmetic operations (including square root). One instruction of particular use in hypercube routing is Find First One, which finds the bit position of the first 1 in a word, via a right-to-left scan. Using a 10-MHz clock, nonarithmetic instructions can be executed at about 2 MIPS, with single-precision floating-point operating at 0.5 MFLOPS, and double-precision at 0.3 MFLOPS. (These performance figures assume that register-to-register operations predominate.) A 32-byte instruction cache allows loops of up to 16 bytes to be executed directly from the cache. The node processor has a vectored interrupt facility, and generates different interrupts to indicate program exceptions such as numerical overflow or address faults, software debugging commands such as breakpoint and trace, I/O signals such as input ready, and hardware errors such as correctable or uncorrectable memory errors.

Pin and silicon space limitations forced a number of design compromises in the selection of the width of various system data paths. The node memory supplies data in 16-bit halfwords, plus an extra byte containing ECC check bits. The processor performs single-error correction and double-error detection (SECDED) on all memory words, generating an interrupt in case of an error. This is an example of a situation where the pin limitations affect performance, for it requires two memory fetches to obtain a full 32-bit word. It also increases the number of memory chips required, since the SECDED code used for 32-bit data could be supplied by five RAM chips organized as 32K $\times$ 8 bits, if such chips were available.

Communication with other nodes is performed via asynchronous DMA operations over 22 bit-serial I/O lines. The I/O lines are paired into 11 bidirectional channels, which permit formation of a 10-dimensional hypercube, and also allow one connection to an I/O board. Each node-to-node channel operates at 10 MHz with parity check, yielding a data transfer rate of about 1 Mbyte/sec per channel in each direction. A channel has two 32-bit write-only registers associated with it: an address register for the message buffer location in the node RAM, and a count register indicating the number of bytes left to send or receive. There is also a ready flag and an interrupt enable flag for each channel. Once a send or receive operation has been initiated by a processor checking its flags and setting the appropriate registers, the processor can continue with other operations while the DMA channel completes the internode communication operation. Interrupts can be used to signal when a channel is ready for a new operation. For general applications, this requires less processor overhead than would occur in a polling communication protocol. An interrupt is also generated if there is a channel overrun, which can occur only on an input operation if more than 9 channels are transmitting data into the node. To reduce DMA activity, a broadcasting feature is supported which transmits the same data word along an arbitrary set of output channels in a single DMA operation.

The NCUBE/ten's I/O boards provide the connections between the hypercube and the external world. Each system must have at least one host board, and may have as many as eight. The host board uses an Intel 80286 to run the operating system, and has 4 Mbytes of RAM used as a shared memory by the various processors on the host board. It has support for a variety of different peripherals, including eight ASCII-standard terminals, four SMD disks (which can currently be as large as 500 Mbytes), and three Intel iSBX connectors that can accept daughter boards for functions such as graphics control or networking. Miscellaneous functions found on the host board include a real-time clock, and temperature sensors for automatic shutdown on overheating. Besides the host board, other I/O boards currently available are a graphics board with a 2K $\times$ 1K $\times$ 8-bit frame buffer, an intersystem board to connect two NCUBE systems, and an open system board with about 75% of the board left for custom design.

A distinguishing feature of the I/O boards in the NCUBE/ten is the fact that each has 128 bidirectional channels directly connected to a subcube of the hypercube; see Fig. 4. This permits extremely high I/O data-transfer rates into the hypercube enabling, for example, a single I/O board to transfer 1024 $\times$ 1024 $\times$ 8-bit images at video rates (30 frames per second). To accomplish this, each I/O board contains 16 NCUBE processors chips, each of which serves as an I/O processor and is connected to eight nodes in the main hypercube. Like the hypercube node processors, an I/O processor has a 128-Kbyte RAM which occupies a fixed slot in the 80286 host's 4-Mbyte memory space. An input operation from the outside world, e.g., a disk read, is performed by first transferring the input data to the host's 4-Mbyte memory. Then the data is transferred through the DMA channels of the I/O processors directly to the target hypercube nodes. Output operations are handled in a similar fashion. In addition to sharing access to the host's memory, the 16 I/O processors on each I/O board are interconnected as two disjoint 3-dimensional cubes, (this disjointness occurs because each node has only 11 bidirectional channels.) Note that in a maximum-configuration NCUBE/ten system, the hypercube nodes do not have to redistribute I/O data to other nodes. This is not the case in smaller NCUBE systems where fewer channels are available for external I/O operations.

An NCUBE system has from one to eight I/O boards (at least one of which must be a host board), from one to 16 processor boards, and attached peripheral devices. All the I/O and processor boards of a fully configured system, along with their fans and power supplies, fit into a single enclosure that is less than 3′ on each side. A full-sized system dissipates about 8 kW, and can be housed in a standard air-conditioned environment. A peripheral enclosure is about 3′ $\times$ 2′ $\times$ 3′ and contains a 65-Mbyte cartridge tape drive and up to four disk drives. A minimal standalone NCUBE system consists of one host board and one processor board containing a 6-dimensional hypercube, and can handle up to 8 user terminals. By adding a second processor board, one obtains a 7-dimensional hypercube. Since the operating system can allocate subcubes of arbitrary size, it is possible to have a number of processor boards which do not form a complete hypercube. For example, three boards provide a 7-dimensional and a 6-dimensional hypercube, which could also be allocated as three 6-dimensional hypercubes or as numerous smaller hypercubes. A full-sized system (Fig. 4) contains a 10-dimensional hypercube (which explains the "ten" in NCUBE/ten). The 1024 processors of such a system together have a potential instruction execution rate of about 2 billion instructions/second, or about 500 MFLOPS. The total amount of memory in the nodes is 128 Mbytes. If

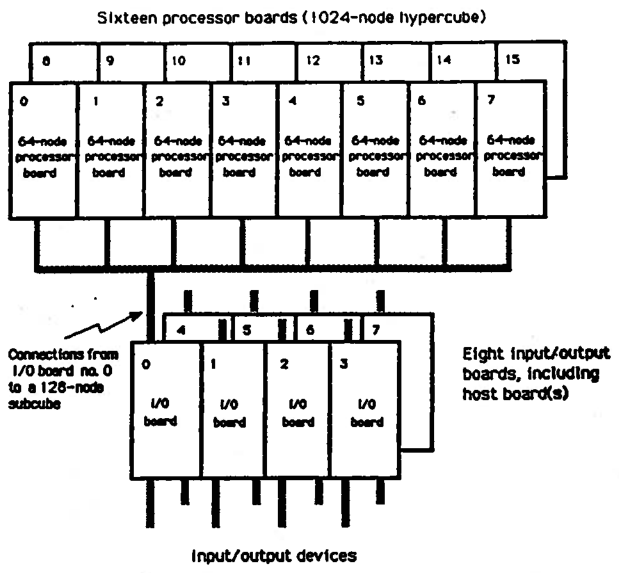
Sixteen processor boards (1024-node hypercube)
8
9
10
11
12
13
14
15
0
1
2
3
4
5
6
7
64-node processor board
Connections from I/O board no. 0 to a 128-node subcube
4
5
6
7
0
1
2
3
I/O board
Eight input/output boards, including host board(s)
Input/output devices

Fig. 4. Maximum-configuration NCUBE/ten system.

all of the I/O boards are host boards, it is possible to support 64 terminals, and provide as many as 16 billion bytes of storage. A host board can provide input or output at up to 90 Mbytes/sec, giving a system input or output rate of about 720 Mbytes/sec.

Figure 5 summarizes the results of some performance experiments designed by D. Winsor at the University of Michigan, which compare the NCUBE node processor to two other representative CPU's with floating-point hardware: the Intel 80286 (the NCUBE host processor served for this) and Digital Equipment Corp.'s VAX-11/780. The measurements were made with the NCUBE node and host processors running under 8 MHz clocks. Extrapolated figures for the 10 MHz version of the NCUBE node processor now nearing production are also given, assuming no wait states. Two widely used synthetic benchmark programs were employed in this study, the Dhrystone and the Whetstone codes [2,18]. The Dhrystone benchmark is intended to represent typical system programming applications and contains no floating-point or vectorizable code. The original Dhrystone Ada code [18] was translated into a FORTRAN 77 version with 32-bit integer arithmetic that attempts to preserve as much of the original program structure as possible. This entailed changes such as replacing Ada records by FORTRAN arrays which produce a substantial performance degradation compared to Dhrystone benchmarks in Ada, Pascal or C. However, any such degradation here appears to apply uniformly to all processors considered, since all were given the same FORTRAN source code and used very similar FORTRAN compilers. The Whetstone benchmark, which aims to represent scientific programs with many floating-point operations, was used in a single-precision FORTRAN 77 version that closely resembles the original ALGOL code [2]. The Dhrystone results in Fig. 5 are reported in "Dhrystones per second," each of which corresponds roughly to one hundred FORTRAN statements executed per second. The Whetstone figures represent the number of hypothetical Whetstone instructions executed per second. It can be concluded from the data of Fig. 5 that the NCUBE node processor is quite fast and fully meets its performance targets cited above.

4. System Software

The emergence of several commercial hypercube computer has demonstrated the feasibility of constructing low-cost massively parallel machines. The focus of research can now be expected to shift to the issue of how these machines can be programmed effectively. Indeed, the recent report on the Supercomputing Research Center concludes that the absence of appropriate parallel programming languages and software tools is the single biggest impediment to the successful use of parallel machines [1]. The operating system is also a major design issue, since memory management and interprocessor communication are critical to the functioning of the programming languages. Three software issues need to be considered. The first is the operating system that is used for developing application programs for the hypercube. The second is the operating system that provides run-time support for application programs running on the hypercube nodes. The third is the set of application languages to be used.

Processor	FORTRAN Dhrystones/sec	FORTRAN Whetstones/sec
NCUBE node processor at 8 MHz	999	381,000
NCUBE node processor at 10 MHz (est.)	1,249	476,000
Intel 80286 (NCUBE host) at 8 MHz with 80287 floating-point co-processor	510	101,000
DEC VAX-11/780 with floating-point accelerator	741	426,000

Fig. 5. Summary of processor benchmark results.

An attractive choice for a development operating system that provides the kind of environment associated with a "programmer's workbench" is UNIX. Unfortunately, there are two different versions of UNIX (System V and bsd 4.2), and a large number of lesser-known variants. This leaves the system designer with something of a dilemma: on the one hand UNIX offers a proven development environment that is widely known; on the other hand a UNIX standard has yet to emerge. The solution chosen by NCUBE was to develop a UNIX-like operating system called AXIS [7] that embodies the features common to the major UNIX dialects. Subsequent change or additions can be readily made to AXIS when a true UNIX standard is agreed upon. There are two features of AXIS that we shall elaborate on here because they are pertinent to the management of a very large hypercube. The first is the ability to share files, and the second is the way in which the main cube array is managed.

AXIS runs on the 80286 host processor that acts as the CPU for each I/O board. (Recall that up to eight I/O subsystems can be accommodated in a 1024-processor NCUBE/ten). It provides the large number of utilities for editing, debugging and file management that one has come to expect in a UNIX-like operating system. Consistent with the UNIX philosophy, the file system is the most prominent feature of AXIS, and almost all of the system resources are treated as files. Massively parallel systems require high I/O bandwidth if they are to be useful for applications that are not simply computation-intensive. This problem of I/O management was not foreseen in the earlier generation of massively parallel machines, and has proved to be a great limitation [10]. The ability to incorporate up to eight I/O subsystems in the NCUBE/ten is intended to avoid this problem. However, it introduces the potential for eight separate file systems. To avoid this, AXIS provides the capability to organize the eight file systems as one distributed file system; AXIS further allows complete systems to be networked through iSBX connections to provide a single multiuser file system. The principal mechanism for doing this is the device directory pointer (ddir). ddirs are items that can be placed in a file directory. Instead of representing a file name, they are pointers to disk drives. Each disk drive has a unique device identifier, which includes a *system number,* an I/O *board number,* and a *drive number*. Within a disk drive, the files are organized as a typical UNIX tree. ddirs can be placed in the root directory of a disk to point to the root of the other drives. This permits file sharing across all physically connected NCUBE systems. Figure 6 illustrates how the directory structure for two host boards, each having two physical disk drives, might be organized, if logical disk0 and disk4 are shared systemwide. Typically, the device directory of a physical disk0 (called "//") will contain the following:

1. The name of another directory which acts as the real root of the file structures on disk0 ("cb0" on host 0)
2. ddir's for the other drives connected to the same host
3. ddir's of disk directories on any other host board in the system
4. ddir's for disks on any other physically connected NCUBE system. Since *system number* is one of the components of a ddir, it can refer to disks on other NCUBE systems.

AXIS manages a hypercube of node processors as a device, which is simply one type of file. It can be opened, closed, written to, and read from as if it were a normal file. AXIS permits users to allocate subcubes that have the appropriate size for their application. Thus, one or two large problems, or several small problems may share the hypercube. This flexibility greatly increases the system efficiency, and gives a hypercube supercomputer a significant advantage over conventional supercomputers. Partitioning the main hypercube into subcubes is simplified by the fact that each subcube is protected from access by any other subcube.

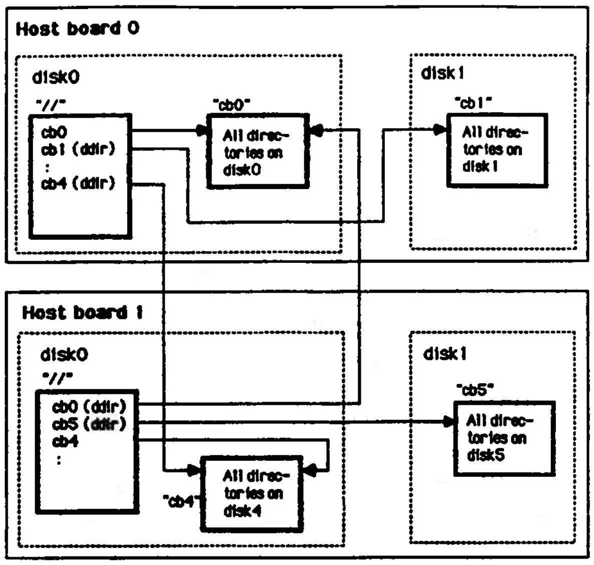

Fig. 6. Distributed files on the NCUBE/ten.

VERTEX, the operating system for the hypercube node processors, is a small nucleus (less than 4K bytes) that is resident in each of the NCUBE/ten nodes. Its primary function is to provide communication between the nodes, in the form of send and receive functions that transfer messages between any two nodes in the hypercube. The node processor has instructions that are used as primitives in the VERTEX communication calls, **nwrite** and **nread**, which implement the internode send and receive functions, respectively. The messages transferred by **nwrite** and **nread** are arrays of bytes having four associated attributes: source, destination, length and type. The first two attributes are numbers in the range 0 to 1023, and indicate the logical nodes being used for source and destination. The length attribute is the number of bytes in the message; messages as long as 64K bytes are supported. Finally, the type attribute can be used to distinguish messages, and so permit their selective reception at a destination node.

The subroutine **nwrite** may be represented as

nwrite *(length, messages, dest, type, status, error)*

where *length* is length of the outgoing message in bytes, *message* is the name of the buffer from which the message is to be taken, *dest* is the logical number of the node in hypercube that is to receive the message, *type* is the type number of the message, *status* indicates when the message has left the buffer, i.e. when the buffer is reusable, and *error* is an error code. Message transmission breaks the message into packets of 512 bytes (or some other user-defined size), and sends them to the destination node using the following routing algorithm. Assume that in an n-dimensional cube, the logical number of the source node is $s_n s_{n-1} \dots s_2 s_1$ and that of the destination is $d_n d_{n-1} \dots d_2 d_1$. The bit-wise exclusive-or $x_n x_{n-1} \dots x_2 x_1$, of the two numbers is formed as follows: $x_i = s_i \oplus d_i$ for $i = 1, \dots, n$. The values of the x_i's are used to control the routing process. Those values of i for which $x_i = 1$ indicate the dimensions that must be traversed to transfer a message from source to destination. The routing algorithm was chosen for its simplicity; however, as noted by Valiant [17], there is a potential for congestion in some situations. He defines an alternative routing algorithm that avoids congestion by routing each message to a randomly chosen node; from there the message is forwarded to its originally intended destination. The randomization assures that message congestion at nodes will be dispersed. Unfortunately, Valiant's router does not perform as well as the straightforward algorithm in many routine parallel processing tasks, and its more complex implementation requirements discouraged use of it in the initial NCUBE/ten design. Future insights into the behavior of parallel algorithms may change this, however.

In addition to determining the routing path, VERTEX must perform the store-and-forward function at each node along the path. At the destination node the message is placed in a queue that is allocated from a heap of 20K bytes. The receive function, which can be represented by

nread *(length, message, source, type, status, error)*

looks for the first message from *source* of type *type* in the input queue, and copies it to buffer *message*. Don't care conditions are indicated for *source* or *type* by setting these parameters to -1. This allows the next message from a particular source to be received regardless of type, it allows the next message of a particular type to be received from any source, and it allows the next message of any type from any source to be received. Messages with negative types other than -1 are system messages for VERTEX and are used for process control at a node, e.g., for node program debugging. In summary, the calls **nwrite** and **nread** provide a fast internode message communication mechanism. The main contributers to this speed are the machine instructions provided explicitly for internode communication, and the fact that messages enter nodes through DMA channels. Measurement of the internode communication performance of the NCUBE system is presently under way.

The current NCUBE/ten application languages, apart from the node and host assembly languages, are FORTRAN 77 and C. The choice of FORTRAN 77 and C was made because the computer is targeted for a user community interested primarily in scientific problems; this group has traditionally programmed in FORTRAN. Compilers for other languages, including Occam, are presently being developed. The programming model adopted for the initial set of languages (FORTRAN and C) is a simple extension of the conventional uniprocessor model. Each node is treated as a separate processor. No symbols are shared between nodes: the naming scope is contained within a node. Values of variables are shared by calls to the VERTEX subroutines **nwrite** and **nread**.

5. Conclusion

Hypercube architectures are well suited to implementing massively parallel supercomputers, given the constraints imposed by current technology. They offer an unusually good combination of high node connectivity, software flexibility, and system reliability. The NCUBE/ten is an example of a new generation of low-cost and compact hypercube machines with the capability of supercomputing performance. Unlike earlier machines, it exploits the inherent homogeneity of the hypercube to provide a multiuser UNIX-like programming environment, along with support for extremely high I/O data-transmission rates.

Acknowledgement

The University of Michigan portion of the work reported here was supported in part by the Office of Naval Research under Contract No. N00014 85 K 0531.

References

[1] Anon: Report of the Summer Workshop on Parallel Algorithms and Architectures for the Supercomputing Research Center, Aug. 1985.

[2] H.J. Curnow and B.A. Weichman: "A synthetic benchmark," *Computer Jour.* vol. 19, pp. 43-49, Feb. 1976.

[3] G. Fox: "The performance of the Caltech hypercube in scientific calculations," Caltech Report CALT-68-1298, April 1985.

[4] F. Harary: *Graph Theory,* Addison Wesley, Reading, Mass., 1969.

[5] W.D. Hillis: *The Connection Machine,* Cambridge, Mass., MIT Press, 1985.

[6] INMOS Corp: *Transputer Reference Manual,* Colorado Springs, 1985.

[7] NCUBE Corp.: *NCUBE Handbook,* version 1.0, Beaverton, Ore., Apr. 1986.

[8] M.C. Pease: "The indirect binary n-cube microprocessor array," *IEEE Trans. on Computers,* vol. C-26, pp. 458-473, May 1977.

[9] J.C. Peterson et al.: "The Mark III hypercube-ensemble concurrent processor," *Proc. Int'l Conf. on Parallel Processing,* pp. 71-73, Aug. 1985.

[10] J.P. Potter (ed): *The Massively Parallel Processor,* Cambridge, Mass., MIT Press, 1985.

[11] F.P. Preparata and J. Vuillemin: "The cube-connected cycles: a versatile network for parallel computation," *Communic. of the ACM,* vol. 24, pp. 300-309, May 1981.

[12] C.L. Seitz: "The Cosmic Cube," *Communic. of the ACM,* vol. 28, pp.22-33, Jan. 1985.

[13] J.S. Squire and S.M. Palais: "Physical and logical design of a highly parallel computer," Tech. Note, Dept. of Elec. Engin., Univ. of Michigan, Oct. 1962.

[14] J.S. Squire and S.M. Palais: "Programming and design considerations for a highly parallel computer," *Proc. Spring Joint Computer Conf.,* pp.395-400, 1963.

[15] H. Sullivan and T R. Bashkow: "A large scale, homogeneous, fully distributed parallel machine, I," *Proc. Computer Architecture Symp.,* pp. 105-117, 1977.

[16[H. Sullivan, T.R. Bashkow and D. Klappholz: "A large scale, homogeneous, fully distributed parallel machine, II," *Proc. Computer Architecture Symp.,* pp. 118-124, 1977.

[17] L. G. Valiant: "A scheme for parallel communication," *SIAM Jour. of Computing,* vol. 11, pp.350-361, May 1982.

[18] R.P. Weicker: "Dhrystone: a synthetic systems programming benchmark," *Communic. ACM,* vol. 27, pp. 1013-1030, Oct. 1984.