N91-23026

HYPERSWITCH COMMUNICATION NETWORK

J. Peterson, M. Pniel, E. Upchurch

Center for Space Microelectronics Technology
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, California 91109

Abstract: The Hyperswitch Communication Network (HCN) is a large-scale parallel computer prototype being developed at JPL and in collaborations with several large computer companies. These companies are planning on building commercial versions of the HCN computer. The HCN computer being designed is a message-passing multiple instruction multiple data (MIMD) computer, and offers significant advantages in price-performance ratio, reliability/availability, and manufacturing over traditional uniprocessors and bus based multiprocessors. The design of the HCN operating system is a uniquely flexible environment that combines both parallel processing and distributed processing. This programming paradigm can achieve a balance among the following competing factors: performance in processing and communications, user-friendliness, and fault tolerance. The prototype is being designed to accommodate a maximum of 64 state-of-the-art microprocessors. A full configuration will provide up to 2.6 GIP, 1.2 GFLOPS, 4 Gbytes of total node memory, 700 Mbytes/second I/O rate and 4 Gbytes/second processor-to-processor communication rate. This communication architecture extends the application of parallel systems to supercomputer problems that place heavy demand on the system for high bandwidth, low latency, and non-local communication. Hence, the classification of the HCN concept as being distributed supercomputing. This paper describes the HCN system, and reviews the performance/cost analysis and other competing factors within the system design.

The research described in this paper was performed by the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology, and jointly sponsored by SDIO/IST and the United States Air Force, Electronics Systems Division through an agreement with the National Aeronautics and Space Administration.

1. Applications: The need to solve more complex problems is outpacing the ability of the world's fastest computers to solve the required applications within acceptable time periods. At the same time, even with the continuing advances in microelectronics technology, it is becoming increasingly more difficult to design and build powerful computers. The demand for increasing capability, higher performance and fault-tolerance are continually being placed on computers. Several application examples are given below.

1.1 Space Flight Operations: The ground based command and control operation system has played a crucial role in JPL and NASA's success. Traditional mainframe computers have been employed for science and engineering applications in the past, providing centralized processing and data management resources for each project. The demand on computation and data handling capabilities multiplies as the complexity of spacecraft and their operation grows; the computational demands is also exacerbated when operations considers the concurrent multiple missions environment. The addition of more computers appears to be the best solution today and the present Space Flight Operations Center (SFOC) system is comprised of about one hundred conventional workstations networked together.

The following Flight Missions: Magellan, Galileo, Ulysses, Mars Observer, TOPEX, and CRAF/Cassini are expected to be operating together in the next decade; EOS, Lunar Observer, Rover, and another half dozen missions are proposed to be supported by JPL. The current Flight Operations network will eventually become "unsteerable" assuming even a factor of four increase in physical count. What is needed is a ground data system that will provide for more accurate and timely data processing that is both informative to the user and cost effective to the project.

1.2 Consolidated Command Center: Military applications comprises information management systems (consolidated command center) to support needs in terms of planning, decision making, and fault diagnosis. As with the JPL needs, ground data systems are needed to help cope with the increasingly complex problems in the logistics of supply and support, as well as with strategic and tactical planning. Although the precise functional and performance requirements of such a consolidated command center are of necessity evolving, certain basic, generic command center requirements include: survivability which places a geographical distribution requirement on the system; high level of fault tolerance; multi-level security; flexible and high bandwidth communications and networking; interfaces with a wide variety of machines; scalability in order to match performance requirements and dynamic reconfigurability to respond to variable workloads; high performance database management; supercomputer class floating point computation speed. Many of these requirements are also needed for the JPL Flight Operations applications. Therefore, these generic requirements are the driving force behind the HCN system described herein.

2. SYSTEM OVERVIEW:

The HCN computer is a loosely coupled MIMD system with distributed local memories attached to multiple processor nodes, see Figure 1. The interconnect topology is a hypercube network used with a hyperswitch[1] message routing element in each node. Message passing is the major communication method among the computer nodes in the HCN computer shown in Figure 1. The HCN Message-routing method demonstrates a highly fault-tolerant capability, while providing an adaptive routing hardware based algorithm with very low message latency. With a very low communication overhead, parallelism is potentially profitable. This is because the programmer seeking maximum performance is strongly tempted to partition a problem into the finest possible granularity to create the maximum amount of parallelism. Therefore, in a message-based parallel computer, the performance of fine granularity computation depends crucially on the rate of message exchange. Fine granularity is a system which effectively supports processes transmitting short messages between code blocks that are less than several hundred instructions in length.

2.1 Hardware: Each node comprises one or more state-of-the-art Motorola microprocessors, and is expected to provide from 50 to 300 MIPS per node with

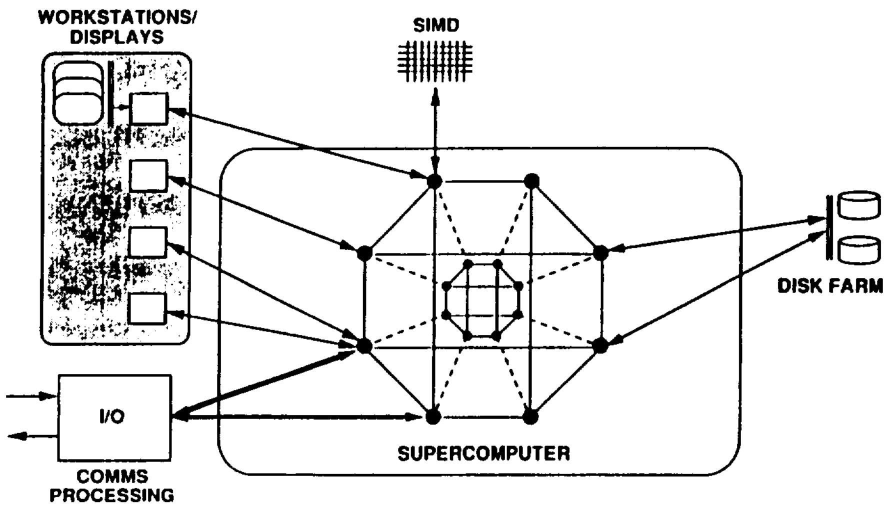

Figure 1, Hyperswitch Communication Network prototype

comparable floating-point performance. The prototype machine as a whole will comprise a total of 32-nodes, at least two microprocessors per node.

The system of links, communication, and application processors thus supplies an homogeneous, MIMD supercomputing resource, while allowing for expansibility and attachment of special-purpose processors. The partitioning of the system into communication, application, and special-purpose devices is reflected in the software: the communication system is completely hidden by the operating system software which presents a distributed object-oriented view. Applications run on the application processors and utilize the attached special-purpose processors in a transparent, fault tolerant fashion. The class/object view which hides the communication system also hides the attached devices without limiting their use.

2.2 Software: The class/object paradigm provides for intra-process (intra-program) communications. Global, named communications links, such as might be needed for a distributed DBMS, are easily implemented. The distributed operating system view is shown in Figure 2 as compared to a current workstation class networking view. We also expect fault-tolerance facilities to be supplied through the object system. By refining an object's simple send method to be an atomic multicast, we set the stage for shadowed processes virtually transparent to the user.

The programming environment will evolve noticeably in ten years. We see powerful debugging and monitoring tools developed which provide facilities equal to or better than those provided by the tools currently available for developing sequential programs on workstations.

3. Architecture: The architecture of the HCN is shown in Figure 3. This architecture is based on the new Motorola M88K microprocessor, a custom message processor and the JPL custom hyperswitch communication chips. The HCN architecture is designed to support applications that require both fault tolerance and high-performance. This architecture extends the application of parallel computers systems to large-scale problems that place heavy demands on the communication network for high-bandwidth, low latency, and non-local communication. The overall performance of systems composed of many tightly coupled processes, such as data searching, sorting, graphics, information processing, etc. depends largely on both the efficiency of communication between nodes and the efficiency of fault recovery. All node functions support this structure.

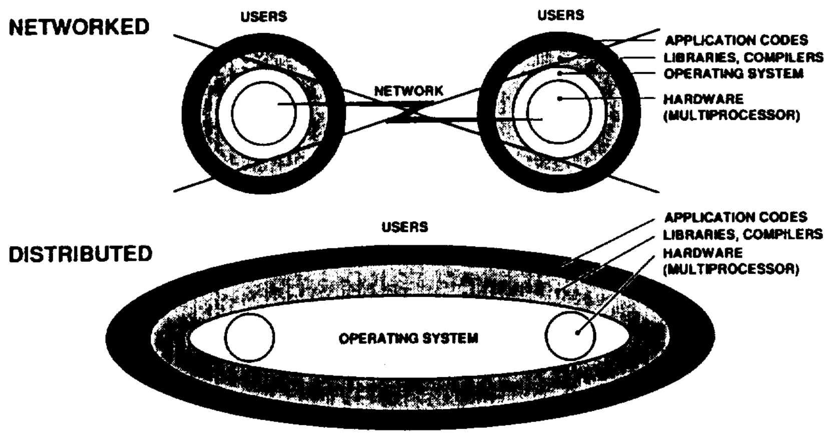

Figure 2, HCN parallel processor operating system with distributed multicomputer operating system support

3.1 Computer Node: The node processor structure shown in Figure 3 consists of two M88K units on one system bus (IMbus). Each unit contains one M88K microprocessor and four M88K cache and memory management devices. These devices include high-speed memory caching, two-level demand-paged memory management, and support for shared-memory multiprocessing. The M88K is a high-speed reduced instruction set computer (RISC) microprocessor. One M88K unit can be configured as the master CPU and the other M88K unit as a checker. This master/checker configuration contains comparator circuits which examine internal and external state of all active output signals. If a mismatch occurs on any output, then an error signal is asserted, the node hardware recognizes the fault and the operating system software reorganizes to eliminate a faulting node by logically enabling a redundant node. In addition to using the master/checking mode for fault-tolerant applications, the two M88K units can be used as independent microprocessors for increased performance (shared node memory multiprocessing).

Message processing latency is reduced by directly executing messages with the custom message processor using special microcode and hardwired logic. The message processor provide support for macro primitive loading and execution, message error checking, message transmission, message broadcasting, process-to-process synchronization, and message receive buffering. The message processor is also configured in a master/checker configuration for fault recognizing and recovery. Also, processor registers are used to save the message transfer control contents when other communication interrupts occur. The custom message processor is designed to field the communication interrupts and handle all the ordinary communication events.

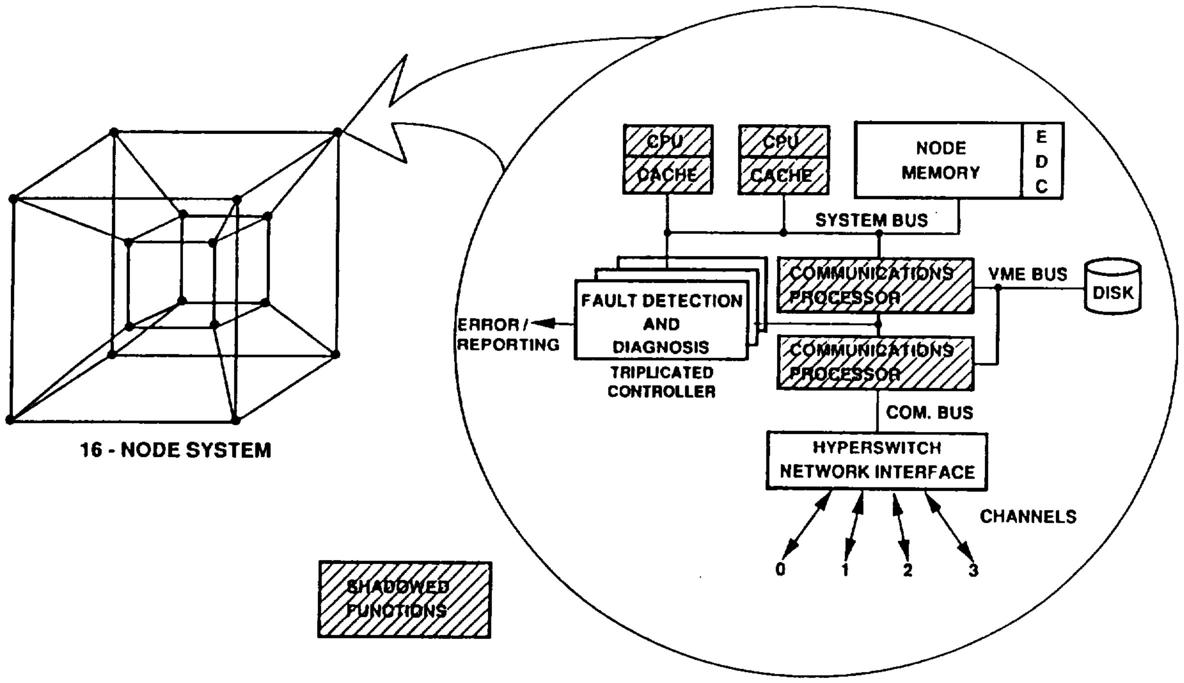

Figure 3, HCN node processor structure

In early hypercube systems message routing latency was in the hundreds of microseconds, but with the development of the hyperswitch communication chips and the message processor, latency has been reduced to a few microseconds. With a hypercube interconnection network, any two nodes that are not directly connected by a link must have their message connected by intervening nodes. But, within a hyperswitch communication network there is virtually no performance degradation when message are sent from one end of the network to the other end. The tested hyperswitch chips use an informed heuristic search algorithm, which can automatically avoid congested or faulty links based on its previous congested experience. Therefore, a message does not wait for a busy link because the hyperswitch network tries to route the message through noncongested or fault-free links. The I/O links are two 200Mbits/s bit serial channels, one for data input and one for data output. One of the log n I/O node links can be selected as a fiber optic channels for long-haul communications. Multiple fault detection and recognition is built into the hyperswitch communication chips. This allows dynamic recovery software to reorganize around a faulting channel or node to restore normal operation.

4. Operating System: The HCN-based operating system (OS) will be a balance among the following competing factors: performance in processing and communications, user-friendliness, and fault tolerance. We can make best use of our resources by adopting existing operating system code wherever possible, and by building a system that supports modern programming paradigms. Figure 4 shows the user environment and concurrency support available from the OS. The primary programming paradigm to be supported is an object-oriented model, in which each object is resident on some node of the machine or distributed over several nodes for performance. Since no single programming paradigm is appropriate for all applications, a second view will be supported: processes and messages. The processes referred to here are either UNIX-style processes or lightweight tasks. While this view is closer to the view given by the JPL Mark III[2], a principal difference will be a marked reduction in the "hypercube" view. The user will be less aware of the cube-based communications than was the case with the Mark III.

4.1 Concurrent Object-Oriented Programming: The HCN operating system supports an object-oriented concurrent programming paradigm. Concurrent object-oriented programming is a methodology in which the system to be constructed is modelled as a collection of concurrently executable program models called *objects*. This powerful paradigm in which the HCN is to be written exploits parallelism both in the architecture and application.

4.2 Programming Environment: C++ is chosen as the primary programming paradigm for the HCN because it can serve two purposes: (1) its compatibility with C makes it a language close to the machine so that all important aspects of a machine are handled simply and efficiently in a way that is reasonably obvious to the programmer. For example, the user creates an object and specifies where the object is to be placed. All subsequent manipulations of that object are done in the usual C++ fashion with no reference to the object's location. (2) the object-oriented features in C++ makes it a language close to the problem to be solved so that the concepts of a solution can be expressed directly and concisely. C++ provides constructs to express class/subclass hierarchies, type abstraction and inheritance. Extensions are added to C++ to support parallel processing, such as remote object creation and concurrent message passing using futures.

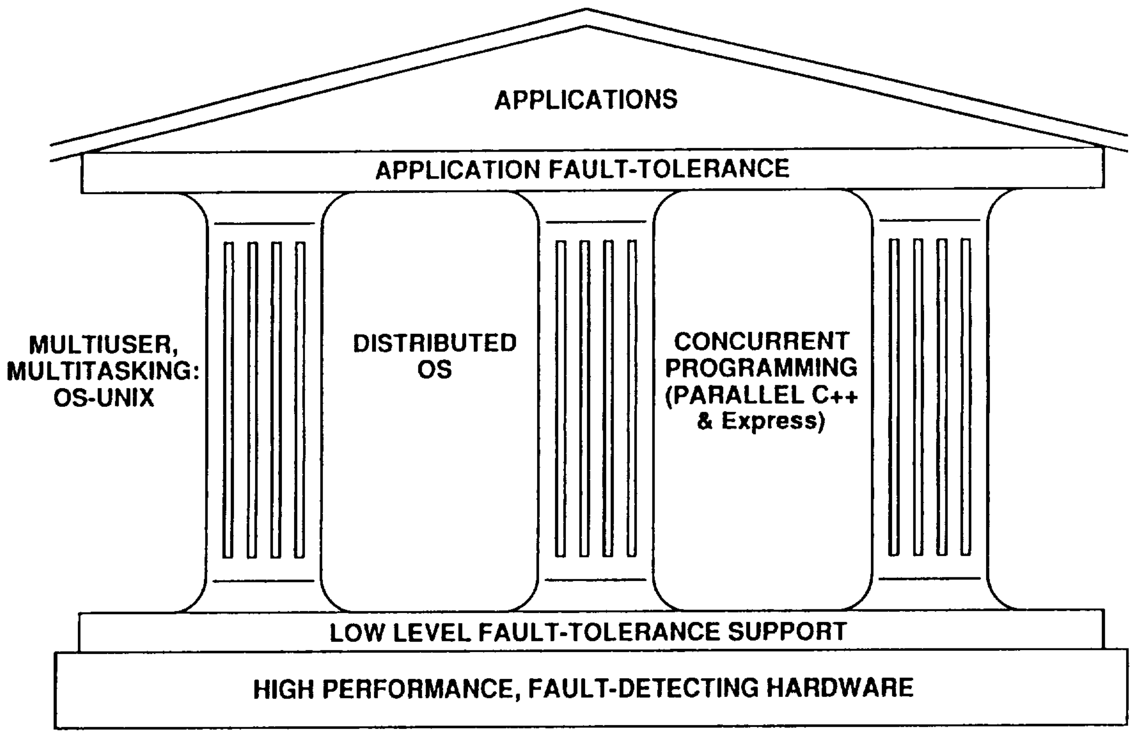
APPLICATIONS
APPLICATION FAULT-TOLERANCE
MULTIUSER, MULTITASKING: OS-UNIX
DISTRIBUTED OS
CONCURRENT PROGRAMMING (PARALLEL C++ & Express)
LOW LEVEL FAULT-TOLERANCE SUPPORT
HIGH PERFORMANCE, FAULT-DETECTING HARDWARE

Figure 4, HCN user environment and concurrency support

4.3 Concurrent Process-Oriented Programming: The process-oriented programming paradigm is a more traditional way to program an application on parallel machines --- a set of sequential processes cooperatively solve a problem by exchanging information via message passing. It has shown that the process model is a powerful programming model for a distributed-memory multiprocessor. Therefore, HCN OS will also support the process model in addition to the Object-Oriented Model. A Unix-compatible distributed operating system, such as Mach[3] or Chorus[4] is under consideration. The advantage for Unix compatibility is that much existing software can be ported to HCN easily, therefore, the application development effort will be significantly reduced. Mach is a multiprocessor operating system kernel developed at Carnegie-Mellon University. In addition to binary compatibility with Berkeley's UNIX 4.3, it also provides facilities for supporting shared-memory or distributed-memory multiprocessors, a new robust virtual memory design and a capability-based interprocess communication facility. The capability-based design and the virtual memory design in Mach can be enhanced to support mandatory access control. By supporting both Object-oriented and Process-oriented programming paradigms with Unix compatibility, fault tolerance, and multi-level security, the HCN OS is capable of serving many different types of applications.

4.4 Process Management: Multitasking will be fundamental to the operation of each node, as well as memory management and memory protection. This is for several reasons, including performance with an asynchronous communications system, multi-user time-sharing of the HCN, and more flexible programming for the user.

The HCN OS process management will be similar to that of Mach. The kernel will support the concept of *threads* (lightweight tasks) which allows the construction of multi-threaded tasks. Such tasks can contain multiple execution paths, all of which can be active concurrently. Threads of a single task can execute concurrently, each in a separate physical processing element. Threads may be created, terminated, suspended and resumed with HCN OS primitives that are much faster than corresponding *fork/exec* 's of UNIX.

4.5 Memory Management: The HCN OS will provide memory management, including virtual memory, similar to that of Mach. The kernel performs memory management at a node where physical memory is treated as a cache for the contents of virtual memory objects. In Mach each virtual memory object is managed by a *pager*. Such pagers could be used to allow memory sharing across a loosely-coupled or distributed configuration.

4.6 Message System: Two HCN OS message management systems will be supported. One is similar to that of ES-Kit. ES-Kit[5] which is an operating system kernel developed at MCC to support distributed, object-oriented execution in extended C++. The other is Express[6] which is a message management system that provides a portable platform on which parallel programs and applications can be built. Therefore, applications built on other concurrent computers using Express can easily port to the HCN computer. Both systems will be employed as the bases of the HCN Operating System. Efforts will be devoted in evaluating the feasibility of adding other parallel constructs, such as Distributed Objects and Multiple Threads. All communication between nodes is by messages. Therefore, both systems must provide communication services without sacrificing performance. Express offers a well understood programming model and is backwards compatible with existing applications. Whereas, ES-Kit offers a much more sophisticated development tool and a strong basis for experimental developments, such as fault-tolerant parallel extensions.

4.7 Programming Tools: A C++ source-level debugger aware of tasks and remote objects, and performance-monitoring tools for visualizing program behavior will be available. The emphasis will be on graphic tools and a simulation environment for the debugging of application code. Graphical tools such as a hierarchical diagram of classes and instances, for example, greatly increase program reliability and programmer productivity.

4.8 Distributed File System: HCN OS supports transparent remote file access whether the file resides on a disk attached to a remote HCN node or on a Workstation connected to the HCN network. The file system and directory structure is Unix compatible. For example, facilities are provided to mount/dismount file systems, and for file transfers between different disk drives.

5. Fault-Tolerance:

The HCN is a set of homogeneous processing elements interconnected by a high bandwidth network. These processing elements are connected to a heterogeneous set of data sources and sinks, including: workstations; graphic displays; disks; and special purpose processors. In order to make the entire HCN a fault-tolerant system, we need to assure fault-tolerant operation in the following three components:

1. The homogeneous processing element has to be fault-tolerant.
2. The communication network has to be fault-tolerant.
3. The interface to the heterogeneous external sources or sinks must be fault-tolerant.

The fault tolerance design of HCN should be able to survive one or more failures falling into any of the above three categories. In the prototype effort, HCN is used as the homogeneous processor network. In the following paragraphs, we will discuss the fault-tolerance design in the HCN with respect to the above.

1. Each HCN node has built-in self-checking hardware consisting of dual Motorola 88000 CPUs for error detection, and error detection circuitry for node memory and system buses. An exception is signalled to the operating system when an inconsistency is detected by the hardware. The OS then initializes the damage assessment program and error recovery program to identify the type and location of the fault, and resumes the whole system to a safe state. In a distributed system, a single node failure cannot be isolated from the rest of the system. Therefore, a global recovery mechanism has to be employed to synchronize and reconfigure the system. The common checkpoint/rollback recovery technique has been identified as inadequate for a message-passing distributed system due to the so-called "domino effect". Instead, user processes are duplicated in two different nodes and processes are synchronized actively by messaging calls. When the primary process is faulty, the backup process will resume the primary's position with minimum recovery delay. This fault tolerance capability is transparent to the user.

2. The Hyperswitch can detect channel errors by two levels of parity check. The adaptive routing algorithm built into hardware can then bypass faulty links and route a message through. It can also recover transient errors occurring in data transmission by automatic retry. Also, self-timing

hyperswitch channels can determine the data rate locally, no system-wide clock is needed. However, the Hyperswitch has limited hardware support for message broadcasting, not to mention atomic broadcasting. Moreover, it does not search exhaustively all the possible routes, and thus may not be able to find a route successfully in the presence of faulty links. The HCN operating system has to perform the following functions to augment the fault tolerance abilities of the hardware:

— To be able to send point-to-point messages in spite of faulty links or nodes in the hyperswitch network.

— To be able to tolerate up to (n/2) link faults, where n is the dimension of the cube. In other words, a point-to-point message can be routed between any two pairs of nodes if there exists less than or equal to (n/2) faulty links.

— Should be able to find a feasible minimal path when the hyperswitch fails to find an optimal route when faulty links exist.

— To be able to broadcast message to the entire cube or to an arbitrary set of nodes

— The broadcast message must either be received by all the non-faulty nodes in the recipient group or none of them (i.e., atomic broadcasting)

— To be able to use a fault-tolerant broadcasting algorithm to bypass faulty nodes and links in building minimal spanning trees.

3. When an HCN node is connected to an external device via a channel, say, a VME interface, either the node, the channel, or the external device may cause single-point failure to the entire system. Therefore, all the three components have to be protected from failure with redundancy. In addition, a fault-tolerant interface between the HCN node and the attached devices/processes has to be built to perform error recovery for the external device. For a device without self error-checking capability, triple module redundancy may be adopted and a voting mechanism has to be included in the interface software/hardware. For a device with self checking capability, a duplicate device is necessary for backup purpose. The physical channels between the node and the external device should also be duplicated to protect the system from single-point failure on the node that is attached to the external device.

6. Cost/Performance Tradeoffs: With today's microelectronic devices the cost of fast devices tends to grow faster than the performance benefit of the increased device speed. Hence, the cost per unit of computing power tends to be greater for high-end machines than for low-end machines, although this trend is technology dependent and could change over time. The relative performances and cost ranges of four classes of commercial computers[7,8] are plotted in Figure 5. The estimated performance and cost ranges of the HCN is also shown. As you can see the low-cost technology of the HCN is an opportunity to create a cost-effective high-performance system by combining slow-speed microprocessors. As stated in Section 2., the cost advantage of using low-cost technology is balanced by the degradation in efficiency that inevitably occurs as the number of processors increases.

Therefore, Communication efficiency and hardware connectivity are the major concerns in the choice of a cost-effective message-passing computer architecture.

To address these concerns, system modeling is used to scientifically explore cost/performance tradeoffs of the HCN-based system[9]. The basic modeling approach is part of a design methodology that has been called performance engineering and uses hierarchical modeling with models expressed by extended directed graphs. High level models are used to evaluate design tradeoffs which includes applications software and operating systems as well as communications hardware overheads. The model can be driven either by synthetic scaled workloads or applications traces.

Some of the design questions being explored with the model include: optimal mapping of application functions to hypercube nodes with and without system faults; sensitivity to message/packet length and network topology; operating systems overhead for various message protocols and data checkpoints; effects of adaptive routing with high volume traffic and bursty traffic.

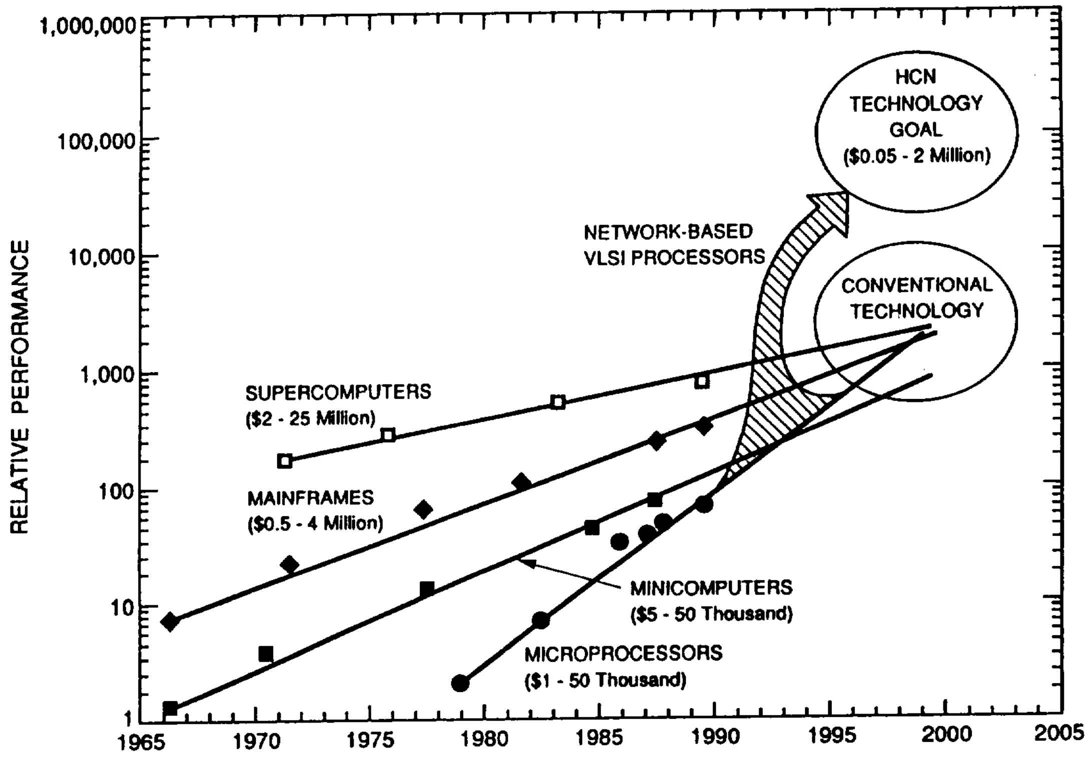

Figure 5, Computer systems cost/performance as a function of trends

7. Conclusion: An important part of this work included establishing a strategy for how long-lived systems should be designed and constructed. In particular, one should view the HCN as an ongoing and continuing design -- the HCN is never complete in the sense that newer and better technology continues to appear, and it must be able to take advantage of that technology. Furthermore, the users of the HCN systems are also changing and improving its utilization -- that is, new applications are encountered and new responses to those applications must be devised and implemented. Therefore, as work gets underway on the HCN, it is important to be designing the next step. Of course, the next step is not a complete replacement, but is an evolution of its fundamental components. By the time several steps of that evolution have occurred, the system may be quite different from its original form, but better and more adapted to the problems that it solves.

Given the rapid pace of technology, a design that reaches ten years into the future can serve best by being flexible -- able to incorporate modification, extension, and, basically, to evolve. In such flexibility there is strength.

References:

1. Chow, E., Peterson, J., Grunwald, D, and Reed, D., "Hyperswitch Network For The Hypercube Computer", Proc. 15th Conf. On Computer Architecture, May 1988.

2. Peterson, J., Tuazon, J., Pniel, M., Lieberman, D., "The MarkIII Hypercube Ensemble Concurrent Computer", Proc. of the 1985 International Conf. on Parallel Processing.

3. Acceta, M., and et. al. , "Mach: A New Kernel Foundation For UNIX Development", Computer Science Department, Carnegie-Mellon University, May 1986.

4. Rozier, M, et. al., "CHORUS Distributed Operating System", Technical Report CS/TR-88-7.6, Chorus Systemes, November 1988.

5. Leddy, W. and Smith, S., "The Design of the Experimental System Kernel" MCC TR# ACA-ESP-089-89, March 1989.

6. ParaSoft Inc., Express Users Manual, Pasadena, Calif., 1990.

7. Hwang and Degroot, Parallel Processing for Supercomputers & Artificial Intelligence, McGraw-Hill, 1989.

8. Patterson and Hennessy, Computer Architecture - A Quantitive Approach, Morgan Kaufmann, 1990.

9. Upchurch, E. and Neuse, D., "Modeling Hypercube Communications Network", Proc. Summer Simulation Conf., Austin, July 1989.