Dudley Knox Library

411 Dyer Road – Monterey, CA 93943-5101
Phone (831) 656-7735; Fax (831) 656-2842;
OCLC Symbol: AD#; Ariel 131.120.51.241; ill@nps.edu

Borrower: IAT

Lending String: ANL,*AD#,WYU,CGU,INU

Patron: DEPT; STATUS; Chabala, Greg

Journal Title: The Third Conference on Hypercube Concurrent Computers and Applications /

Volume: 1 **Issue:**
Month/Year: 1988 **Pages:** 38 - 60, 384 - 390, 843 - 846

Article Author: Conference on Hypercube Multiprocessors (3rd ; 1988 ; Pasadena, Calif.)

Article Title: R. Arlauskas, P. Close, S. F. Nugent, P. Pierce, CORPORATE Intel; iPSC/2 system; a second generation hypercube, The iPSC/2 node architecture, The iPS

Imprint: New York, N.Y. ; Association for Computi

ILL Number: 14155334

Call #: Electronic

Location:

ARIEL
Charge
Maxcost: $15IFM

Shipping Address:
ILL, Lovejoy Library
Southern Illinois University Edwardsville
30 Hairpin Drive
Edwardsville, IL 62026-1063

Fax: 618-650-2381
Ariel: 146.163.157.74
Email:

ARIEL Document Problem Notification

If there is a problem with this ARIEL document, please notify us within **72 hours** of receipt. Simply complete this form and return to us via ARIEL or FAX.

Date: _____ Your OCLC Symbol: _____ ILL/Transaction Number: _____

Please check ☑ and complete all that apply:

- ☐ Missing Page Number(s): _____
- ☐ Edges Cut Off Page Number(s): _____
- ☐ Difficult/Unable to Read Page Number(s): _____
- ☐ Other: _____

Thank you!

iPSC®/2 System: A Second Generation Hypercube

Ramune Arlauskas

Intel Scientific Computers

Introduction

This paper examines Intel's second generation hypercube, the iPSC®/2 system. It is characterized by major advancements in performance, system configuration flexibility and useability. The iPSC/2 development is the result of close work with the iPSC User base and the broader concurrent research community.

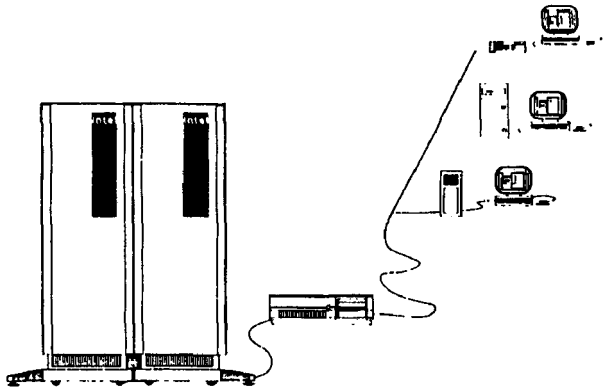
Based on experience with the first generation iPSC system, a key design criteria for the second generation machine was to provide a better balance of communications performance with the processing power.

The second design criteria was to permit the system to rapidly change with the evolution of technology. Through system configuaration flexibility, the user can grow and change his system as technology advances and as his specific application needs change.

Improved productivity was the third design criteria. Parallel programming tools for the iPSC/2 system needed to substantially improve program development productivity. The tools need to be an extension of the programmer's familiar workstation environment.

iPSC/2 System

The typical iPSC/2 system consists of three basic components.



iPSC®/2 Concurrent Supercomputer

The cube itself which houses the nodes connected in a hypercube topology. The System Resource Manager which serves as the administrator console, hosts the software development tools known as the Concurrent Workbench™ and serves as the gateway to other computer resources and workstations. Finally, the network of user workstations enables the user to develop applications from his familiar environment.

Performance

The iPSC/2 system delivers significant performance improvement. Figure 1 compares two dimensional Fast Fourier transform (FFT) times for the iPSC-VX and iPSC/2-VX system, and Cray X-MP/12.

The 2D FFT is an excellent example of worst case communications traffic in hypercubes. A matrix is mapped onto the nodes in a row orientation. After each node transforms the data in the first dimension, it must exchange different data with every other node. This floods the communication network with messages. In the first generation machine, it takes several seconds to complete the transpose. No floating point computation occurs during this period.

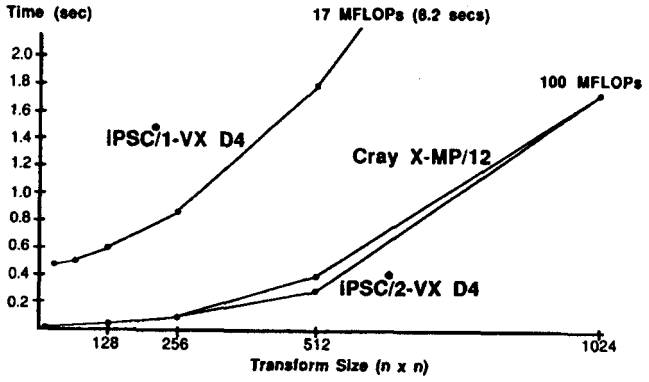


Figure 1: 2D FFT Performance

For a matrix of order 1024, the iPSC/2-VX system realizes a 6x improvement both in FFT solve time and MFLOPs over the first generation machine. Comparing the results against a Cray X-MP/12, the solve time is 50% faster on the iPSC/2 system with comparable MFLOP performance.

Looking at a complete application, NEKTON* is a commercially available fluid dynamics and heat transfer modeling code. Using spectral element techniques, it also requires significant computation and communications power. In figure 2, we see that the iPSC/2-VX realizes a 65% gain over the first generation system and a 30% gain over a Cray X-MP/14. Clearly, the iPSC/2 system delivers supercomputer performance.

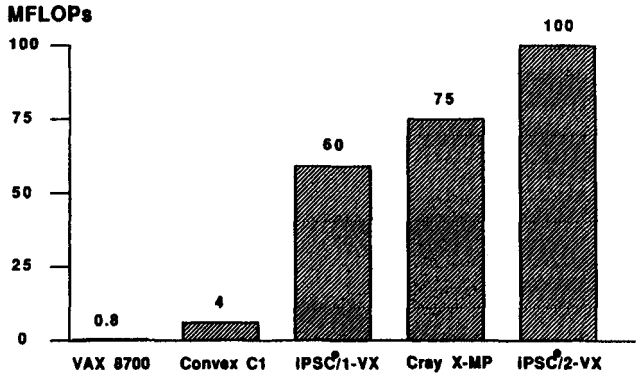


Figure 2: NEKTON Performance

New Communication Network

The outstanding performance results in both examples are due to advancements of the iPSC/2 computation and communication tehcnologies.

Each iPSC/2 node uses the Intel 80386 32 bit microprocessor which delivers 4 MIPs performance. Each node is equipped with the Direct Connect Module™, DCM, for high speed routing of messages between nodes. The collection of DCMs produce the high speed communications network of the iPSC/2.

In the first generation iPSC system, the store and forward technique was used for message passing. Each node in the communications path stored the message in its memory. The processor on each node in that path was also involved in the communications handling and during those periods was not available to do other processing.

The Direct Connnect Network can be thought of as a switching network. When one node wants to communicate to another, a series of switches are closed and the communications path established. Once the path is built, messages proceed at the full hardware speed of 2.8 Mbytes/second. Only the sending and destination processors are involved in the communication. The other processors in the path continue with their normal activities. Since it takes only a few microseconds per node to build the path, the additional overhead for multi-hop communications is insignificant. Single-hop and multi-hop message transmittal is almost uniformly fast.

Complimenting the hardware performance advancements, the node operating system, NX/2, has been rewritten to take complete advantage of the new hardware. It has been designed to support fast, reliable message passing. Unlike the first generation operating system, messages are not packetized. Rather, NX/2 manages message flow control and minimizes deadlock through new message passing protocols and controlled buffering. Messages are transmitted in their entirety after sufficient buffer space is available on the receiving node. For 0 byte messages latency is about 350μs. For long messages full hardware speed can be realized since the entire message is sent at once.

These advancements in the communications technology supported by a streamlined operating system provide a better balance with the computational capabilities of the system. Both the 2D FFT and NEKTON code performance increases are the result of the improved balance.

System Configuration

Know that its constituent technologies undergo constant advancement, the iPSC/2 system has been designed to incorporate improvements over time without extensive redesign.

While the node board forms the basis for this flexibility, every component of the iPSC/2 system has been designed with flexibility as a key design criteria. Figure 3 illustrates the modularity of the iPSC/2 node board.

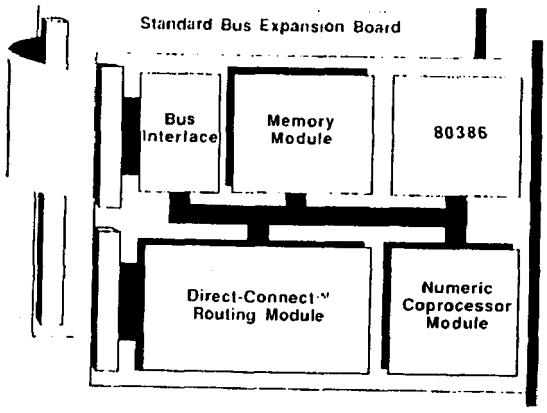


Figure 3: iPSC/2 Node Board

The base platform consists of the processing element, the 80386, and several modules connected by a local bus. The modules include: Memory Module, Communications Module (DCM), and the Numeric Coprocessor Module. Thus as communications, memory and numeric technology evolve, the system is positioned to smoothly incorporate the evolutionary improvements.

The DCM is implemented with CMOS programmable gate array logic. The chips are programmed at system boot time. While the pin-out is etched in the board, the details of the router design can be easily altered.

The Memory module supports 1 to 16 Mbytes of memory per node. The modules are implemented in 1, 4, and 8 Mbyte platforms using standard commercial components and surface mount manufacturing teechnology. 16 Mbytes require stacking two 8 Mbyte boards. These modules were developed for use with other Intel products.

By leveraging from other commercial developments, the iPSC/2 can enjoy not only volume cost reductions, but also quicker time to market.

The Numeric module offers the 80387 Arithmetic Coprocessor standard in the product. 32 bit precision arithmetic executes at 250 KFLOPs with 64 bit precision arithmetic at over 210 KFLOPs. For greater scalar performance, the SX option delivers over a MFLOP peak performance per node and about 650 KFLOPs for 64 bit arithmetic. The SX option uses Weitek 1167 chips and plugs into the 80387 socket. Both SX and the 80387 can co-exist.

To futher extend the modularity, the node has a standard bus interface which allows each node to be coupled with a peripheral standard bus board. Currently the iPSC/2 system supports the iLBX II interface with a Vector Extension board, VX. This board features pipelined vector capabilities with peak performance per node in excess of 6 MFLOPs for double precision arithmetic and 20 MFLOPs for single precision arithmetic.

The NX/2 operating system has been designed in a modular way and implemented in C. As the hardware evolves, the operating system will evolve with it. Also as new software capabilities such as I/O, are added, they can be easily incorporated in the base system.

The most important aspect of the flexibility is that the various capabilities can be mixed to form hybrid systems. For example, within a single cube, one can have nodes with various memory sizes, a mix of scalar and vector exelerator, a mix of standard bus boards, and programs implemented in various languages, complete configuration flexibility for the user's needs.

Software Development Environment

To increase productivity in the development and debugging of concurrent applications, the iPSC/2 system provides a tool set know as the Concurrent Workbench. This tool set offers popular languages including C, Fortran, and Concurrent Common Lisp™ whose compilers generate full 32 bit code. A vectorizing Fortran precompiler, VAST-2, helps the user optimize his application for execution on the iPSC/2-VX system. To compliment VAST*-2, there is an extensive library of vector arithmetic functions. iPSC/2 simulators are available for developing and executing programs on other UNIX*-based machines. A source level debugger, DECON, provides traditional symbolic debugger features and special capabilities for observing, controlling and altering message-passing in programs. With the Concurrent Workbench, concurrent programming is now as sophisticated as sequential programming.

The Unix-based software development environment supports multi-users who can access the Concurrent Workbench from their workstations. For example the iPSC/2 programmer can develop his application on a SUN*-3 workstation. The user has workstation access to the Concurrent Workbench as well as to his workstation specific tools.

The applications interface NX/2 has been simplified through the use of several different levels of message passing protocols. As in the iPSC system, both syncronous and asyncronous calls are supported, however, the type and number of parameters have been simplified. Interrupt driven message passing is also available for greater program sophistication.

Conclusion

The major advancements of the iPSC/2 system have been reviewed. Additional details are available within these proceedings [1,...,7].

New processor, communications and operating system technologies provide a balanced system that delivers supercomputer performance.

Significant advancements in iPSC/2 software development tools increase productivity during program development.

Modularity in the base design of the iPSC/2 system provides the platform for easily incorporating technological advancements in processing, communications, numerics, memory, operating system and program development environments.

In summary, all of these advancements have preserved the users' investments in first generation parallel programming and prepared the iPSC/2 system for commercial application developments.

References:

[1] William L. Bain and Shala Arshi, *Hypersim: Hypercube Simulator for Parallel System Performance Modeling,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[2] Paul Close, *The iPSC/2 Node Archetecture,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[3] Dave Ertel, *Environment for Enhancing iPSC/2 Programmer Productivity,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[4] Steve Nugent, *The iPSC/2 Direct Connect™ Communications Technology,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[5] Wei-min Pan and Victor Jackson, *A Concurrent Debugger of iPSC/2 Programmers,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[6] Paul Pierce, *The NX/2 Operating System,* Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

[7] Dave Scott, *Application Performance on the iPSC/2* , Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

The iPSC®/2 Node Architecture

Paul Close

Intel Scientific Computers

ABSTRACT

Feedback from users of the first generation iPSC® system plus the development and availablility of new VLSI based technologies drove the design of the iPSC/2 node. The new node design was broadly governed by the following goals:

- Provide true 32-bit node architecture and performance.
- Match the communication performance to the computational performance.
- Increase on board memory capacity by using new RAM technology.
- Employ a modular functional elements to easily incorporate future technology.
- Allow plug compatibility with existing iPSC systems, including interface to co-processors.
- Ensure software compatibility for existing iPSC applications.

This paper describes how the iPSC/2 node achieved these goals by leveraging Intel's VLSI expertise and products, surface mount, CMOS and gate array technologies, and small daughter boards to implement modular subsystems.

1. Features of the iPSC®/2 Node

Increased performance, more memory, greater configurablity, and backward compatibility were the key design goals of the iPSC/2 node.

Major features of the iPSC/2 node are:

--A high performance 32-bit, 'off the shelf' commercial microprocessor as the core CPU (Intel 80386), paired with 64Kbytes of high speed instruction and data cache for 0-wait state operation, resulting in a single node with the performance of most supermini-computers (see Section 5 for measured performance data).

--Improved communication to computational performance ratio by increasing communication bandwidth, and eliminating the store and forward technique of the original iPSC system.

--Up to 16 Mbytes of dynamic RAM can be local to each node.

--Modular design: Numeric co-processors, memory, and the routing logic are all added to the node base assembly via modules, allowing users to choose configurations to fit their needs.

--Form factor compatible the original iPSC node, including the interface to the iPSC/VX vector co-processor.

--Software compatible with most application code written for the original iPSC machine, which was based on the Intel 80286.

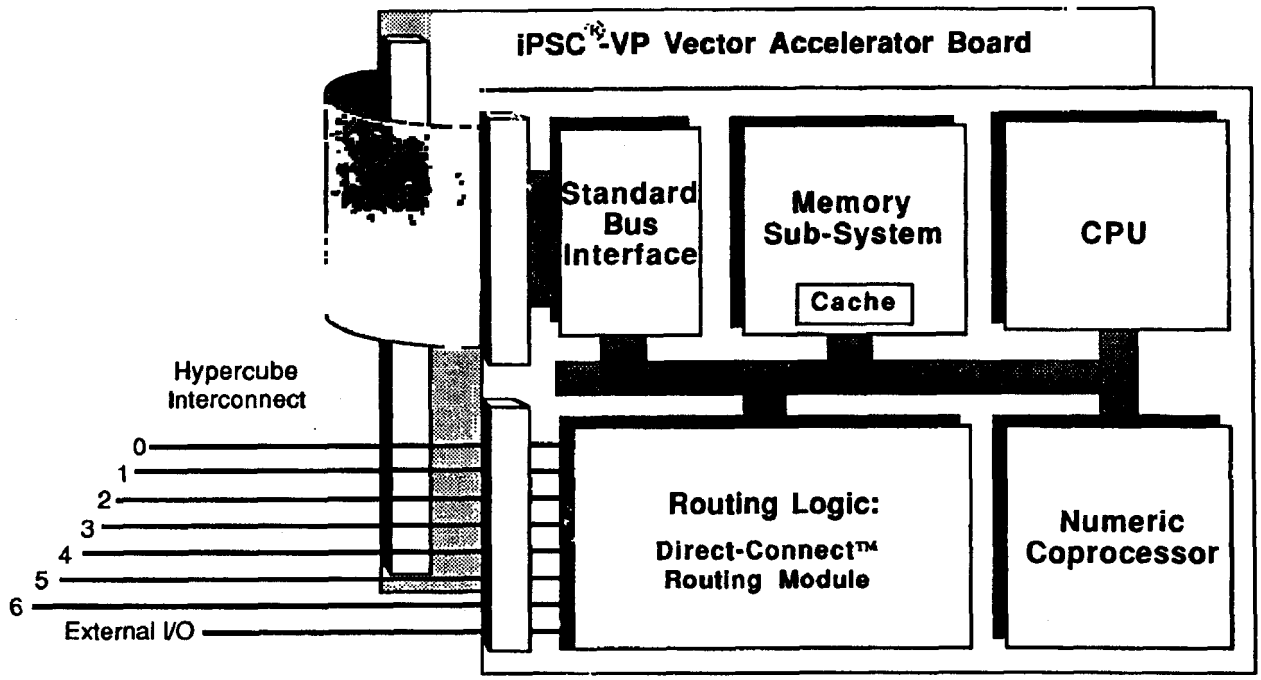


Figure 1. iPSC®/2 Node Block Diagram

2. iPSC/2 Node Description

Refer to Figure 1 for a diagram of the main functional blocks of the iPSC/2 node. Each of the blocks is described in detail in the following sections.

2.1. CPU

The main processor of the iPSC/2 node is the Intel 80386 microcomputer. The 16MHz version used in the node has a rating of 4 MIPS. The 80386 was selected not only for it's performance, but also because it is completely software compatible with the 80286 processor used in the original iPSC machine.

The 80386 has separate 32-bit data and address paths. A 32-bit memory access can be completed in only two clock cycles, enabling the bus to sustain a throughput of 32 megabytes/sec.

Pipelined architecture enables the 80386 to perform instruction fetching, decoding, execution, and memory management functions in parallel. Because the 80386 pre-fetches instructions and queues them internally, instruction fetch and decode times are absorbed in the pipeline; the processor rarely has to wait for an instruction fetch.

Pipelining is not unusual in modern microprocessor architecture; however, including the memory management unit (MMU) in the on-chip pipeline is somewhat unique to the 80386. The integrated memory management and protection mechanism translates logical addresses to physical addresses and enforces the rules necessary for maintaining task integrity in a multi-tasking environment. By performing the memory management on chip, the 80386 eliminates the serious access delays typical of implimentations that use off-chip methods.

The 80386 can respond to an interrupt in as little as 3.5uS. The overhead for task switching is not much longer. In this case the 80386 can save the state of one task (all registers), load the state of another task (all registers, even segment and paging registers), and resume execution.

The iPSC/2 80386 is complimented and tightly coupled with several support components:

--82510 USART

--8259 interrupt controllers

--8254 timer

--82258 Advanced DMA controller

The 82510 USART interfaces the node to the diagnostic link. this link is RS-422 based, and provides the node with a low speed, high reliability 'back end' communication path for initialization. It operates independently from the routing logic. The 82258 interfaces the node memory system to the routing logic, and is explained in Section 2.4.

2.2 On Board Numeric Co-processor

The iPSC/2 node currently supports three numeric options. Two of those are intended for scalar operations, and reside on the node. The third is the VP co-processor board, which attaches to the node via the Standard Bus Interface. The VP board is intended for vector operations.

The on-board numeric operations both attach via the same 121 pin socket. They are:

--An Intel 80387 Numeric Co-processor

--An Intel SX Scalar Extension Module

The 80837 is a companion processor of the 80386, and extends the 80386 architecture with floating point, extended integer, and BCD data types. It uses an 80-bit internal architecture, and fully conforms to the ANSI/IEEE floating point standard. It is logically mapped into the I/O address space of the 80386.

The SX module is based on the Weitek 1167 numeric chip set. It is designed to be a higher performance clone of the 80387, giving roughly 2-3 times the performance. The SX module is memory mapped, occupying address range C0000000-C1FFFFFFH.

The SX module has a socket for the 80387, and since the two options do not electrically or mechanically conflict, it is possible for both to be used simultaneously.

2.3 Memory Subsystem

The memory subsystem consists of three components:

1. Up to 16-Mbytes of Main Memory.

2. 64-Kbytes of Cache RAM.

3. Non-volatile EPROM

2.3.1 Main Memory

Main Memory is attached via Intel's standard MMOX surface mount dynamic RAM, (DRAM) memory module.

Memory modules exist in configurations of 1, 4, and 8-Mbytes. At least one memory module but at most two memory modules may be installed on a node. Two 8 Mbyte modules must be stacked for a 16 M byte configuration. With this configuration the adjacent slot in the cube unit is displaced, halving the number of nodes possible in an enclosure.

Eight bits on the memory module, called MSIZE0 through MSIZE7, provide the node with the necessary information to determine the amount of memory installed.

2.3.2 Cache Memory

64-Kbyted of fast static RAM are used as a data and code cache. Up to 16 Mbyte of DRAM can be present, and so only select DRAM locations are shadowed by the cache, based upon most recent access.

When a DRAM location is read that has been cached (i.e. it has a shadow entry in the cache RAM), the data is returned from the cache, resulting in 0 wait state 80386 reads rather than the 2 wait state DRAM read. In such a case, DRAM is not accessed. On writes to DRAM, the cache entry is updated if the DRAM location being written is cached (hence the term 'write-trough'). In this case the cycle is extended and runs with 2 wait states. If a location that is not cached is read from DRAM, it will simultaneously become cached, possibly replacing another location cache entry. A write to a non-cached location leaves the cache contents unaffected.

The cache memory and the DRAM share the same lower sixteen address lines, so the 64-Kbyte of cache can wrap around 256 times within the 16-Mbytes of DRAM. This is avoided by the tag RAMs, which are also addressed by the same lower 16 address lines. The tag RAMs store the upper eight bits of address that determine which one of the 256 possible DRAM locations is cached (16-Mbyte => 24 address bits, 24 - 16 => 8). When a memory access occurs, the lower sixteen address bits select an entry in the Tag RAM.

The tag RAM data is compared to the upper eight bits of the current address. If the compare is successful, the desired memory entry is in cache, if not, the cycle is a miss.

2.3.3 Read Only Memory

64-Kbytes of EPROM are installed on the node. This EPROM contains the node confidence tests (NCTs), as well as the boot loader.

2.4 Routing Logic Interface

2.4.1 Philosophy

The iPSC/2 node was to be as independent of communications technology as possible. This was done to allow future routing technologies to be used with today's node, allowing existing hardware to be re-used, and faster time to market. To this end, a generic interface has been designed to connect the node memory subsystem with the routing logic, that attaches via a surface mount daughter card.

The routing logic interface to the node is a simple 32-bit wide DMA interface which supports transfers between the routing module and node memory. It 'steals' cycles from the 80386 for the transfers, and can burst at a rate of 10.7 Mbytes/sec.

On the iPSC/2 system, the routing logic daughter card is called the Direct-Connect™ module (DCM). Each DCM contains eight serial channels, implementing, together with the back-plane, a hypercube interconnect topology. The DCM is a surface mount board containing thirteen programmable gate arrays.

2.4.2 Balanced Data Rates

The DCM has sixteen channels, eight in and eight out, which connect the node to its eight nearest neighbors. Each DCM channel can sustain a data rate of 2.8mbytes/sec, independent of other channels. The data will only become invasive to a particular node if it is sourced by, or must sink to that node.

Since the node can source or sink the routing logic at a burst rate of 10.7Mbytes/sec, it is nicely balanced with the interface, being roughly twice the 5.6Mbyte/sec rate (2 X 2 8Mbyte/sec) required to keep up with messages coming in and out of the DCM simultaneously.

For more information on Direct-Connect routing, see the paper within these proceedings entitled: "The iPSC/2 Direct Connect Technology", by Steve Nugent [1].

2.4.3 Physical Connections

There are two 96-pin connectors on every node, that plug into the backplane. One is dedicated to communications, the other is dedicated to the standard bus interface. The routing logic interface consists of two 100-pin surface mount connectors which secure to the node board.

Except for power, ground and the diagnostic link bus, all communication pins are left unspecified by the node and are simply connected to the routing logic interface, leaving the personality of the pins up to the routing logic module.

2.4.4 Data Transfers

Data transfers across the routing interface are controlled by an 82258 Advanced DMA controller (ADMA), which arbitrates with the 80386 for the memory system. Two DMA channels are provided, one for node to routing logic transfers, the other for transfers in the opposite direction. While both channels share the same

physical bus, transfers in and out of the node can be interleaved, logically occurring at the same time. The 82258 ADMA controller is quite flexible, able to handle any size transfers, limited only by the amount of memory installed.

Transfers from the network to the node are initiated by the routing logic module. When the routing logic module has data to be transferred into node memory, it requests a routing logic read from the DMA controller by asserting a DMA request signal. The ADMA will reply with a DMA acknowledge, and initiates the cycle for as long as the request is held active.

Transfers from the node to the routing logic are done in similar fashion. The DMA controller asserts a DMA request signal to the routing logic, which will respond with a DMA acknowledge when free to take data.

2.5 Standard Bus Interface

The iPSC/2 node features a standard bus interface which is tightly coupled to the 80386 CPU bus. The intent is to provide a mechanism to attach option boards of popular buses to the iPSC/2 node. The slot adjacent to the node is populated by the option board, possibly via a small bus adapter board, depending on the native bus of the option board.

To attach a companion board to the system, the odd slot adjacent to the node board contains the companion board, and not another node board. As an example, a iPSC-D4/VX machine has 16 nodes, each made up of an iPSC/2 node board and a VP board. The node boards are in the even slots of the chassis, and the VP boards reside in the odd slots.

2.5.1 ILBX™-II INTERFACE

The iPSC/2 node can be configured to directly interface to any Multibus-II board which supports the LBX-II subset. A good example of a use for this bus is the VP Vector Co-Processor.

The LBX-II interface is a subset of the the Intel Multibus® II standard. It is a synchronous bus, which can support as many as six agents, two requesting agents and four replying agents (synonymous with master and slave respectively). The iPSC/2 machine restricts this to one requesting agent, the node, and one replying agent, normally the iPSC/VX vector processor. Such as:

The LBX-II supports 32-, 16- and 8-bit transfers. LBX memory is enabled by a control bit under software control. Once enabled, LBX memory starts at address 8000000H, and uses the upper 8Mbyte space of the node's 16Mbyte space.

The LBX-II interface can burst data up to 8M bytes/sec, but is of course limited by the speed of the attached board.

3 LEDs

In keeping with the monumental importance of LEDs on scientific machines, here lies a brief but detailed description of the node LEDs.

Three LEDs are mounted on the front panel of the node. The colors are: red, green, and yellow. The red and green LEDs are under software control. The amber LED is hardware dedicated to indicate that a numeric co-processor is active.

The red and green LEDs are typically used by the node in two modes:

--Reporting results of the node Confidence Tests upon their invocation by power up or system diagnostics. The red LED to indicates failure, the green indicates all tests passed.

--During normal run time, the node executive NX/2 uses the green LED to indicate normal computation mode, while the red is used to indicate communication to another node is transpiring.

4 Initialization and Reset

Upon power up or reset, the node processor executes on-board diagnostics. If the node passes the NCTs, it enters a firmware resident boot monitor, and receives a download of the node Executive (NX/2) from the Systems Resource Manager (SRM) via the DCM routing logic.

After the node has received it's copy of NX/2, it begins executing it from RAM. NX/2 initializes on-board hardware as required, then enters normal system mode. At this point, NX/2 awaits download of application code from the SRM, again via the Routing Logic Interface.

5 Measured Performance

The performance numbers given here reflect measurements taken on the iPSC/2 Beta release 1.0 software. Intel Scientific Computers expects some increases in performance as message passing software is tuned.

The data reflect three different configurations of the iPSC/2:

1. A single iPSC/2 node with 80387. In this configuration the numeric processing is provided by an Intel 80387.

2. A single iPSC/2 node with the SX Scalar Floating Point module. In this configuration the 80387 is augmented with the SX module. As can be seen, this gives 2- 2.5X the performance of the 80387 configuration.

3. A single iPSC/2 node with the iPSC/VX Vector Co-processor.

Performance numbers for a VAX-11/780 have been included as an industry standard reference.

	iPSC/2 80387	iPSC/2 SX	iPSC/2 VX	VAX-11/780
Dhrystone				
Drystones/Sec.:	7812	7936	NA	1562
Whetstone				
Single Precision KWIPS	1498	3501	NA	1133
Double Precision KWIPS	1299	2206	NA	727
Daxpy				
MFLOPs for N = 1	0.05	0.07	0.06	NA
MFLOPs for N = Infinity	0.14	.0.30	2.60	NA
Linpack				
MFLOPs for 100 X 100	0.16	0.33	1.43	0.14
MFLOPs for 300 X 300	0.16	0.34	2.37	NA
MFLOPs for 1000 X 1000	0.16	0.34	2.50	NA

	iPSC/2 80387	iPSC/2 SX	iPSC/2 VX	VAX-11/780
Livermore Loops				
Mean Node Speed				
DP MFLOPs for N = 17	0.15	0.25	NA	NA
DP MFLOPs for N = 90	0.15	0.26	NA	NA
DP MFLOPs for N = 471	0.15	0.26	NA	NA
Peak Node Speed				
DP MFLOPs for N = 17	0.21	0.63	NA	NA
DP MFLOPs for N = 90	0.18	0.60	NA	NA
DP MFLOPs for N = 471	0.25	0.61	NA	NA

References

[1] Nugent, Steve, *The iPSC/2 Direct-Connect Technology*, Proceedings, 3rd Conference on Hypercube Concurrent Computers and Applications, 1988.

The iPSC®/2 Direct-Connect™ Communications Technology

Steven F. Nugent

Intel Scientific Computers

ABSTRACT

This paper describes the hardware architecture and protocol of the message routing system used in the iPSC®/2 concurrent computer. The Direct-Connect™ router was developed by Intel Scientific Computers to replace the store-and-forward message passing mechanism used in the original iPSC system. The router enhances the performance of the iPSC/2 system by reducing the message passing latency, increasing the node-to-node channel bandwidth and allowing simultaneous bidirectional message traffic between any two nodes. The new communication system has nearly equal performance between any pair of processing nodes, making the network topology more transparent to the user.

The Direct-Connect router is a specialized self-contained hardware module attached to each hypercube node. The router is implemented in CMOS programmable gate-arrays with advanced CMOS buffering. Routers are connected by full-duplex bit-serial channels to form a boolean n-cube network. The router also provides a high performance interface between the node memory bus and the network.

INTRODUCTION

The Direct-Connect router is a hardware controlled message passing system. It forms the basis for the message passing system of the iPSC/2 second generation concurrent computer. When interconnected to form a hypercube network, Direct-Connect routers provide the means to pass messages of arbitrary size between pairs of computational "nodes" which interface to the routers.

The routers form a circuit-switched network that dynamically creates a synchronous path, from a source node to a destination node, which remains open for the duration of a message. The path is composed of a series of channels that form a unique route from the source node to the destination node and may pass through some number of intermediate routers associated with other nodes. When more than one channel constitutes a path, this is referred to as a "multi-hop" route.

Channels are bit-serial and full duplex and connect nearest neighbor nodes. The Direct-Connect router supports connections for eight full duplex channels and can be interconnected to form networks of up

to seven dimension containing 128 nodes. Each of the eight channels is routed independently allowing up to eight messages to be routed simultaneously. One channel per router is dedicated to act as an external path into the network and allows remote devices to access the full routing capabilities of the network. The router communicates with the node over two unidirectional parallel buses.

Routing is based on the e-cube routing algorithm [6,7], which guarantees a deadlock free network. Paths are dynamically constructed for each message prior to its transmission. A complete path is built in a step-by-step process involving arbitrations for additional path segments at each router. The channels that constitute a path are held for the duration of the message. When a destination node is ready to accept a message, transmission begins. A channel is released when the tail of a message passes between the routers connected by that channel.

Direct-Connect routing is a variation of Wormhole routing [1,2] with the primary difference being that with Direct-Connect the message is transmitted after the route has been built. This difference allows the system to operate completely synchronously and eliminates the need for flow control buffering in the intermediate routers of a multi-hop route [4].

Because messages are routed independently by hardware, the iPSC/2 message passing latency is significantly reduced over the original iPSC system. The message passing system employed in the original iPSC hypercube used a store-and-forward packet-switching technique which was typical of first generation hypercube machines. Using this technique, message packets were stored in their entirety and re-transmitted at each intermediate node of a multi-hop path. This led to long message route times and also interrupted the processes of intermediate nodes in the path that were not recipients of the message. The Direct-Connect router allows messages to route through intermediate nodes without interrupting processes on those nodes. Messages encounter minimal delays in routing through intermediate nodes and so the transit time of a multi-hop message is only marginally longer than a single hop. As a result the details of the network topology are virtually invisible to the user. It appears to be a fully interconnected network and consequently the mapping of applications is greatly simplified.

The Direct-Connect router has been implemented using programmable gate-arrays and is fully contained on a 4" by 8" daughter card that attaches to the iPSC/2 node card. This assembly is referred to as the Direct-Connect Module or DCM.

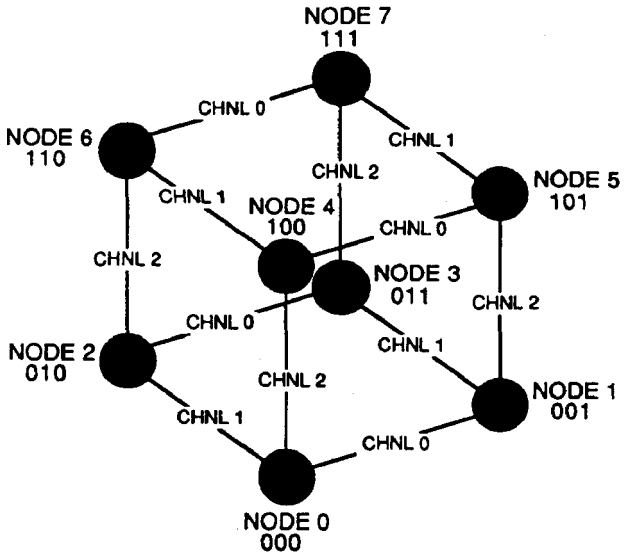
Hypercube Nomenclature

The iPSC/2 system consists of a collection of single board processors or "nodes" interconnected with full-duplex bit-serial channels to form a hypercube. In a hypercube where each node has N nearest neighbor nodes the system is said to have dimension N.

Figure 1 illustrates the channel and node naming convention used in a iPSC/2 hypercube of dimension 3.

The nodes are assigned unique addresses so that the address of any two nearest neighbor nodes differ by one binary digit. The channels that connect nearest neighbors are named for their corresponding dimension. The dimension of a channel between two nodes is determined by taking the binary exclusive--or of the two node

addresses. The bit position that remains a one is the dimension of that channel. For example, the channel connecting nodes 5 and 7 is determined by the exclusive or of 111 and 101. The result is 010 and because the "one" is in bit position one, that channel is in dimension one.



DIMENSION 3 HYPERCUBE

Figure 1

Router Architecture

Message Paths

As shown in figure 2, the router is actually composed of eight independent routing elements, one for each of eight incoming channels (numbered 0-7). These routing elements dynamically create message paths through the DCM's. Each routing element is capable of driving several outgoing channels, one at a time (the connectivity is defined by the e-cube [6,7] routing algorithm). Since more than one routing element may request the same channel simultaneously, an arbitration mechanism for each channel resolves the conflict fairly.

The node interface consists of two unidirectional parallel channels, node source and node sink. All routers may request the node sink channel and likewise the node source channel has access to all outgoing channels.

The channel seven routing element is special and its routing algorithm does not operate like channels 0-6. It acts instead as a repeater for external node source and sink channels. This provides an I/O gateway into and out of the network for remote devices such as disk farms, graphics devices and real time I/O. Node 0 channel seven serves as the host interface.

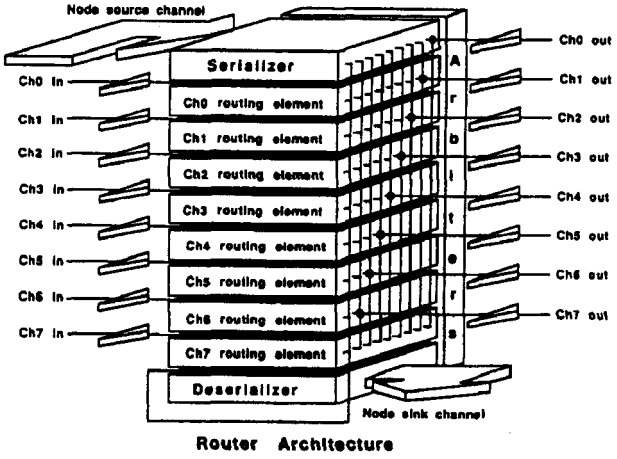


Figure 2

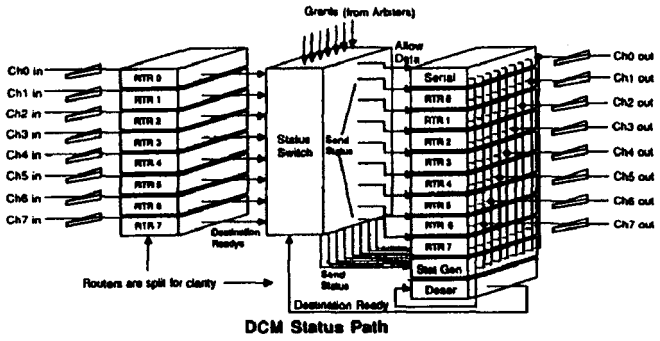
Status Paths

In addition to the primary message paths, there is a secondary path, the status path, that routes status from the destination to the source of each message. The status path is necessary to provide flow control for messages. To pass status between routers, status information is multiplexed onto the channels during message transmission. In the absence of messages, status information is passed continuously.

The status path within a router functionally consists of a Status Switch, Status Generators, and

contained within the routers, DESTINATION READY registers and SEND STATUS logic. As illustrated in figure 3, the organization of the status path allows DESTINATION READY bits received at the routers to be passed to other specific routers to be re-transmitted as SEND STATUS. Each DCM is capable of routing status information for eight simultaneous messages.

The dynamic connectivity of the status switch is controlled by the channel grants from the arbiters. Consequently, the connectivity of the status paths is determined by the current message routing. Using this mechanism the status generated at the message destination can make its way back to the source by passing through intermediate routers along the path in the opposite direction from the message.



DCM Status Path

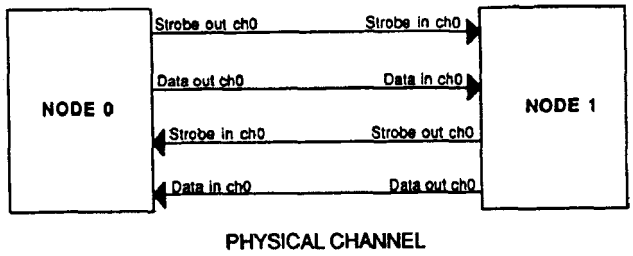
Figure 3

The origin of the ready status is the destination nodes Deserializer which generates this signal based on the reception of a routing probe and the status of the receive FIFO. After passing through any intermediate DCM's the signal arrives at the source DCM Serializer as an ALLOW DATA control signal. Allow data, as the name implies, controls the transmission of data from the source DCM Serializer.

The purpose of the Status Generator is to generate status in the absence of message traffic. It transmits the same SEND STATUS as the routers on all channels that are idle. An idle channel is one that is not currently reserved for a message.

Channel Description

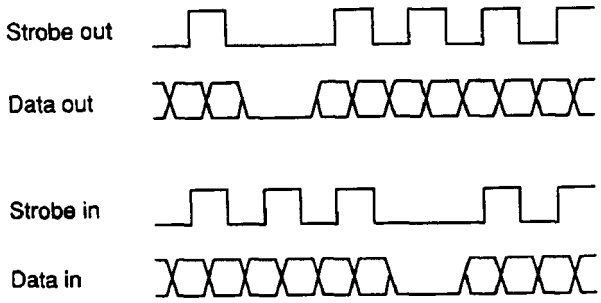
The Direct-Connect channel physically consists of four conductors that connect each of the nearest neighbor nodes. One pair of conductors is driven in only one direction and the other pair only in the opposite direction. The pairs operate independently of each other. Figure 4 illustrates the connectivity between nearest neighbor nodes.



PHYSICAL CHANNEL

Figure 4

Serial data, control and status bits are transferred across the data lines. The strobe lines are used to validate the data lines and also provide a clock source for the subsequent router. Both rising and falling edges of the strobes validate the data lines. Figure 5 illustrates the physical level channel timing.



Channel Timing

Figure 5

The system is clocked using the strobe lines. Each message path is completely synchronous. This means that different messages are not

synchronous to each other but that all intermediate logical elements involved in routing a particular message are synchronous. When message data is routed through a DCM, the associated Strobe also takes the same route. Therefore, once the route is established, the original strobe from the source DCM actually clocks the entire data path from source to destination.

The main advantage of "one-way" channels over full handshake protocols is that the transfer rate can be much higher. Transfer rates of handshake channels suffer because of the latency caused by end to end acknowledgements and the fact that speed degrades as the channels are made physically longer. The use of FIFO buffers at the message destinations and synchronous channels eliminates the need for handshake protocols. Consequently, the throughput is not a function of channel length or acknowledgment delay. It is possible to build a reliable channel that uses wormhole routing techniques without using a handshake protocol.

Channel Protocol

A hardware level of protocol exists between nearest neighbor routers in order to provide a means of passing control and status information. Two status/control bits are passed on a continuous repetitive basis whether or not message transmission is occurring. These bits are END OF MESSAGE (EOM) and READY STATUS (RDY). The EOM bit indicates that the last word of the message has been transmitted and is ignored unless a message is in progress. The RDY bit represents the state of readiness of the destination node of an established path.

Two formats exist to allow the EOM and RDY bits to be interspersed within messages or to be passed in the absence of message traffic. These formats provide a convenient way of identifying the EOM and RDY bits so that they may be stripped off and processed at each router. The status only format is termed the "status nibble" and consists of four bits, the first two being ones indicating a status only transfer and the last two being the RDY bit and a don't care. These status nibbles are repetitively transmitted by all routers in the absence of data transfer. The data format also contains four bits of non-data information. The first two bits are zeros indicating that a message is in progress and the last two are the RDY bit and the EOM bit. The data and status nibble formats are illustrated in figure 7.

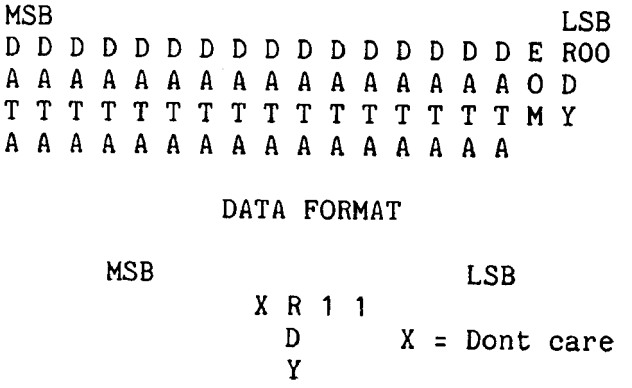


Figure 7

The RDY bits are stored as they are received at each router in "Destination Ready" registers. These form the basis of the message flow control mechanism used in the system.

The reason that there are two "start bits" on both the status and data formats is that the message is processed in two halves in the routers. The odd numbered bits are handled independently from the even numbered bits in the router in order to achieve higher data rates than

would otherwise be possible given performance of the gate arrays.

Because status information is interspersed with message data, the end of message can easily be detected by routers on the fly. This eliminates the need for a message size counter in the routers and thereby removes any limits to maximum message size.

Message Routing

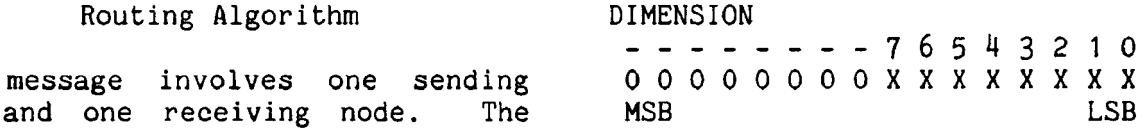
Routing Algorithm

Each message involves one sending node and one receiving node. The routes that messages take through the network are unique between any two nodes. The combination of channels that compose a path are defined by the e-cube routing algorithm [6,7]. Using this algorithm guarantees that no circularities will occur in message routing and thus prevents hardware deadlock from occurring.

The algorithm states that in order to guarantee deadlock freeness, messages in hypercubes can be routed in increasingly higher dimensions until the destination is reached. The channel numbering defined in the Hypercube Nomenclature section corresponds to these dimensions. Paths may consist of increasingly higher numbered channels but may not necessarily be contiguous. Routing to lower numbered consecutive channels is not allowed. For instance, a path may consist of channel 0 - channel 2 - channel 3 which involves the DCM's of nodes 0, 1, 5 and 13. In this case the source DCM is at node 0, the intermediate DCM's are at nodes 1 and 5 and the destination DCM is at node 13.

Routing Operation

A routing operation can be broken into four phases: establishing a path, acknowledgement, message transmission and releasing connections. To initiate the routing of a message, the source node must transfer at a minimum one 32 bit word to the DCM. The low order 16 bits of this first 32 bit word must contain the ROUTING PROBE. The routing probe contains addressing information and is used to establish the connections that make up the path that the message must take. The high order 8 bits of the routing probe must be all zeros as shown in figure 8.



Routing Probe

Figure 8

The value of the routing probe is calculated by taking the exclusive or of the binary address of the destination node with that of the source node. Each bit of the routing probe corresponds to a channel that the message can be routed on.

The first segment of the path is established when the Serializer in the source DCM requests the outgoing channel that corresponds to the lowest order bit set in the routing probe. These requests are arbitrated by the local DCM. The DCM arbitrates between local requestors for the same channel and grants one at a time using a "round robin" arbitration scheme.

When the channel is granted, the routing probe is sent by the source DCM before any message transmission takes place.

For example, if a routing probe is transferred to the DCM in which bit N is the lowest order bit set, channel N will be requested. When the DCM grants channel N, the routing probe will be transmitted to the

intermediate DCM that is the nearest neighbor on channel N.

Upon receiving the routing probe, the intermediate DCM will store it and discard the upper 8 bits of zeros creating a short routing probe. The discarded bits will be reconstructed at the destination DCM. The short routing probe will be passed between intermediate DCMs reserving additional segments of the path. Its format is shown in figure 9.

DIMENSION	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	X
	MSB							LSB

Short Routing Probe

Figure 9

The intermediate DCM's examine bits N+1 to 7 in the short routing probe to determine the lowest order bit that is set. The outgoing channel that corresponds to the bit that is set will be requested and the short routing probe will wait. When the outgoing channel is granted the short routing probe will be transmitted to the next DCM in the predefined path. As illustrated in figure 10, this process will repeat until the routing probe is received by the destination DCM.

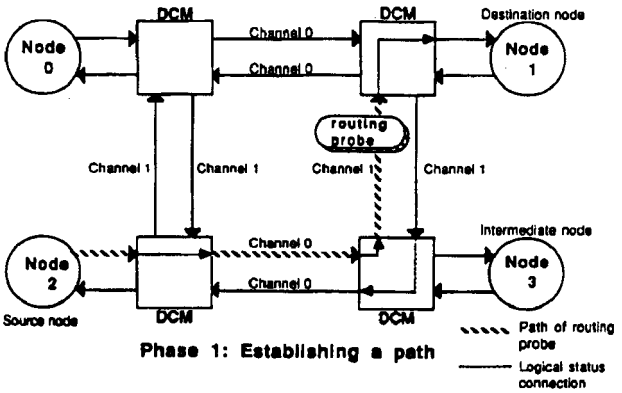


figure 10

After the routing probe is absorbed at the destination, it will be padded with 8 zeros to restore it to its original state. If the destination DCM can accept a message, it will signal an acknowledgement, the RDY bit.

This begins the acknowledgement phase of the routing operation. The acknowledgement phase requires that a deterministic connection be made from the destination DCM back to the source DCM for the purpose of carrying flow control information. This is termed the "status path" and follows the message path exactly but in the opposite direction, from destination to source. The status path is established as a result of forming the message path.

For example, if a message is being routed from CHANNEL 2 IN to CHANNEL 4 OUT at an intermediate DCM, a connection from CHANNEL 4 IN to CHANNEL 2 OUT is made for status propagation. This status connection is not a wired path between nodes like the message path but consists of transferring the status information received on one channel to the sender on another channel.

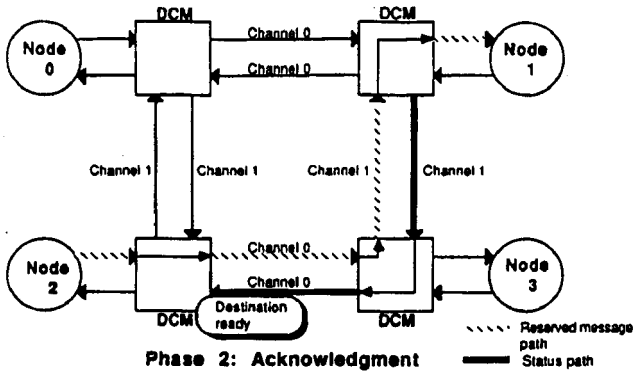


Figure 11

The status path like the message path maintains its connection for the duration of the message. Once the RDY bit is set at the destination of the message, figure 11 illustrates how it propagates over the channels opposing the message path through the intermediate DCMs until it reaches the source DCM.

When the RDY bit finally reaches the source node then transmission of status nibbles will cease and the message transmission phase may begin. The source DCM can transmit data continuously into the network (in the format described in the channel protocol section) until the end of message is sent or a not ready indication is received over the status path. Figure 12 depicts the message transmission phase. The message is not buffered in the intermediate DCMs.

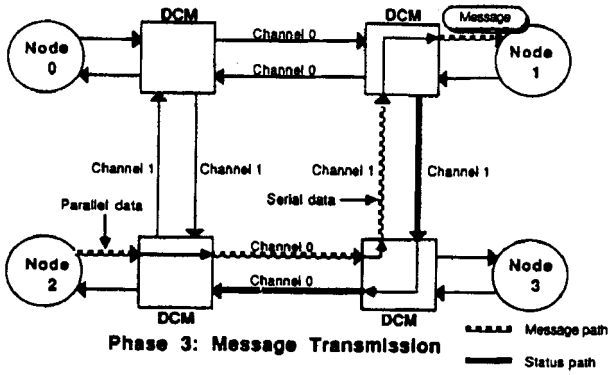


Figure 12

If the source DCM receives a not ready indication from the destination DCM, it will discontinue transmitting data and resume transmitting status nibbles. This is referred to as "throttling" and provides end-to-end flow control for the network. After a message is throttled, status nibbles will flush the message out of the network completely. When RDY is again detected at the source DCM, the transmission of status nibbles will cease and message transmission will resume. The DCM's are able to absorb any data that is in transit on the network or cabling because there are FIFO buffers at the receiving DCM's. Therefore, when a message is throttled, no data bits will remain stored on the network but will be absorbed at the destination DCM.

To complete a routing operation the source DCM appends a checksum word to the message with the EOM bit set in its associated status. The checksum provides a means to verify message integrity in order to detect hardware failures should they occur. As shown in figure 13, the transmission of the checksum word/EOM causes the source DCM to release the outgoing channel reserved for that message. Likewise, at each intermediate DCM in the path, the channels reserved for that message are released on the fly as the checksum is transmitted on. Those channels are then free to be used for other messages.

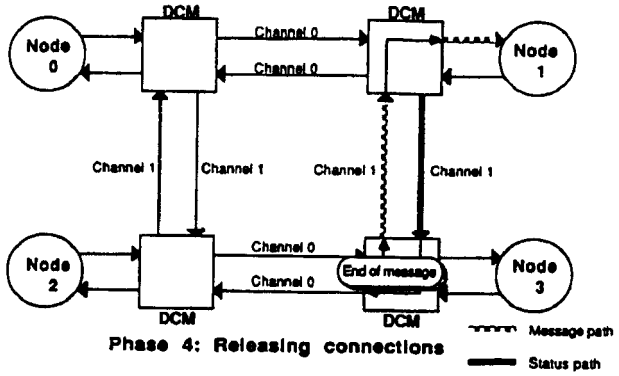


Figure 13

At the destination DCM the reception of the checksum and EOM bit allows the message integrity to be checked and the end of message indication to be passed to the node. Since the checksum is not part of the original message, it is stripped off at the destination DCM and its state stored for further inspection at the destination node.

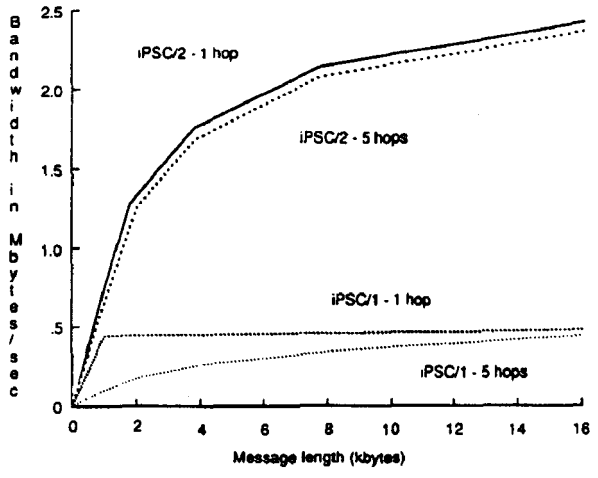
Performance

Direct-Connect routing has both increased hardware bandwidth and reduced latency over the original iPSC computer. To give a qualitative feel for the performance achieved, some actual data points have been plotted using the beta level software and hardware of the iPSC/2 system and data from the first generation iPSC system. A simple program called

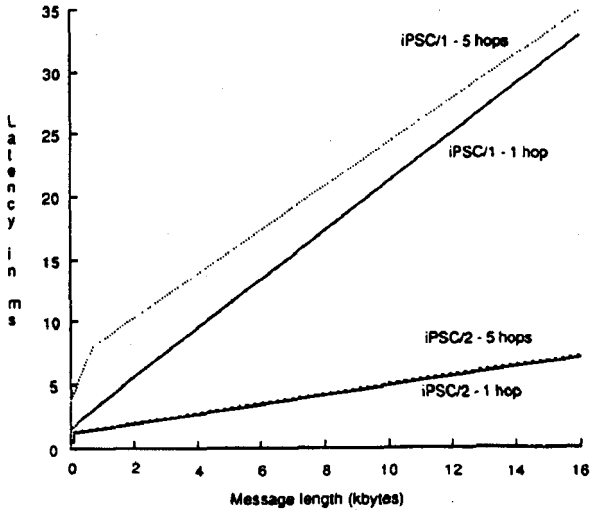
"echo" was used to gather the data in which only two nodes are involved and alternately send messages to each other. No contention effects will be exibited in this example. Overall bandwidth and latency (software, hardware and message transit time between processes) have been plotted as a function of message length for nearest neighbor (1 hop) and multi-hop (5 hop) cases. Since in both the bandwidth and latency plots the primary difference between the single hop and five hop cases is the contribution of the message passing hardware, it becomes obvious that this contribution is small compared to the software contribution. In fact the latency due to hardware is on the order of 5% of that due to software. The latency plot indicates that the iPSC/2 latency is less than one-fifth the latency of its predecessor, the iPSC/1 system.

The most dramatic effect of Direct-Connect routing on the bandwidth, other than a general increase due to faster channels and more efficient protocol, can be seen in the short message portion of the plot. In this region the iPSC/1 system pays a significant penalty for sending a short message many hops whereas it is obvious in the iPSC/2 system case that the effect of single hop versus multi-hop is insignificant on overall bandwidth.

Multi-hop Bandwidth



Multi-hop Messsage Latency



Conclusions

The benefits of Direct-Connect routing technology in hypercubes have been enumerated. It has been shown that the benefits of wormhole routing can be achieved using an implementation that is completely synchronous rather than the self-timed implementations of [2, 3]. It has been shown that networks can be built that can be extended over significant physical distance without the need to change signalling protocol. It can be derived from the performance results that the efficiency of the Direct-Connect technology results in latencies that are insignificant compared to the software necessary to service messages in the iPSC/2 model of medium-grained computing. In other words, for this model of computation, further increases in the performance of the message passing hardware will yield little in terms of system performance. Future computing models, however, such as [5] that reduce software overhead to take advantage of fine-grained decompositions will require even lower hardware latency, as this again becomes a more significant part of the performance equation.

References

[1] C. L. Seitz, et al., The Hypercube Communications Chip, Dept. of Computer Science, California Institute of Technology, Display File 5182, March 1985.

[2] W. J. Dally, A VLSI Architecture for Concurrent Data Structures, Ph.D. Thesis, Department of Computer Science, California Institute of Technology, Technical Report 5209, March 1986.

[3] C. Flaig, VLSI Mesh Routing Systems, Dept. of Computer Science, California Institute of Technology, Technical Report 5241, May 1987.

[4] P. Kermani & L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique", Computer Networks, Vol 3., 1979, pp. 267-286.

[5] W. C. Athas & C. L. Seitz, The Cantor User Report, Version 2.0, Dept. of Computer Science, California Institute of Technology, Technical Report 5232, Jan. 1987.

[6] C. R. Lang Jr., The Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture, Dept. of Computer Science, California Institute of Technology, Technical Report 5014, May 1982.

[7] H. Sullivan & T. R. Bashkow, "A Large Scale Homogeneous Machine", Proc. 4th Annual Symposium on Computer Architecture, 1977, pp. 105-124.

The NX/2 Operating System

Paul Pierce

Intel Scientific Computers

Abstract

NX/2 is the operating system which runs on the nodes of the Intel iPSC®/2 concurrent supercomputer. NX/2 provides all of the standard system services found in the original iPSC node operating system, such as memory management, multiple process control, message passing services, and intertask protection.

This paper focuses on the major node operating system enhancements brought about by two different requirements. First, NX/2 had to support very high speed and high throughput message passing. In this regard, we show how NX/2 was tuned to the 32-bit architecture of the iPSC/2 nodes and the Direct Connect™ technology used to implement the communication subsystem.

Second, NX/2 had to support a more streamlined and flexible set of message passing service calls. In this regard, we describe the new set of message passing system calls and discuss how NX/2 implements them. The calls range from a simple, effective set of synchronous calls to advanced asynchronous calls which allow overlap of message passing and processing as well as interrupt-driven message handling.

1. Introduction

The NX/2 operating system runs on each node of the Intel iPSC/2 concurrent supercomputer. This is a second generation distributed memory, message passing parallel computer based on the Intel 386 microprocessor and the proprietary Direct Connect communications subsystem. It shares with the first generation iPSC computer a microprocessor from the same iAPX 86 family and the hypercube interconnection scheme.

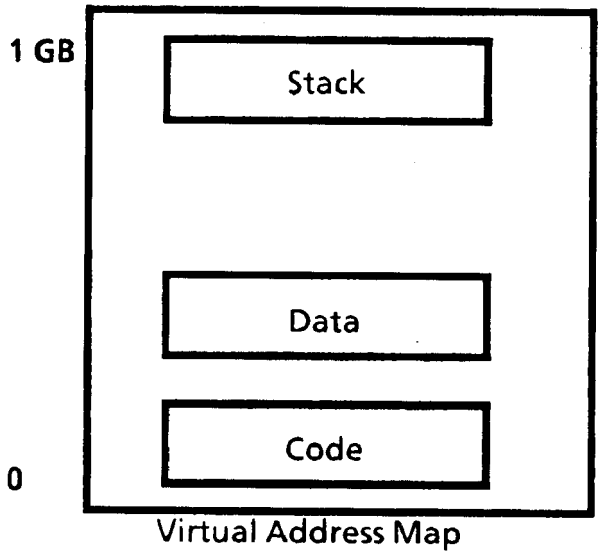
The new operating system also shares the same general approach as the earlier system, oriented toward program-directed message passing with multiple processes per node. However, the details of the process model in the new operating system are quite different due to the 32-bit capability of the processor. The message passing system calls were also revamped to increase programming convenience and flexibility.

2. General Features

The operating system and applications run on the 386 in full 32-bit Native Mode. The system is written almost entirely in C, with the obligatory few parts in assembly language for performance and access to privileged instructions. There are approximately 20,000 lines of C, resulting in 90K bytes of code.

2.1 Addressing

Each application process has a 1 Gigabyte flat virtual address space implemented via the 386 paging hardware. There may be up to 20 processes on each node, each with a separate address space. Code and data may occupy any part of the address space limited only by the available physical memory. NX/2 uses paging to manage the physical memory and individual process virtual address spaces, but does not provide virtual memory, in that every page in use by a process must reside somewhere in physical memory as long as the process exists.



Virtual Address Map

In a typical application, ld (the standard Unix* linker) assigns code addresses around 0 in the virtual address space, and data addresses starting at 400000 hex. The NX/2 loader reads the Unix COFF (Compatible Object File Format) file and places code and initial data in physical memory pages. The loader then maps the pages to the appropriate virtual addresses and allocates a stack at the top of the address space. Additional pages may be mapped to the process if it calls for them via the standard memory allocation routines such as sbrk() and malloc(). Each process has its own address space, but all processes share the same pool of pages in physical memory.

2.2 Numeric Coprocessors

NX/2 also manages the numeric coprocessors (387, SX, or VX) for each process. In the case of the 387 or SX (Weitek 1167), each process has its own coprocessor state. The system saves the old state and

restores the new state when, after a process switch, the new process attempts to access the coprocessor. Since there is a relatively large state for each coprocessor, this technique saves context switch time in the case where only one of several processes is actually doing floating point computations.

The VX vector processor is managed more simply by restricting use to a single process. If an attempt is made to load two processes which need access to the vector processor, the second attempt will fail unless the first process has terminated.

2.3 Direct Connect Interface

NX/2 operates the DMA controller which controls the Direct Connect message passing hardware. For each message sent, the system software realigns the message data to 32-bit boundaries (the message passing hardware is limited to operation with word-aligned data), generates a message header, and sets up the DMA controller for header and data transmission. The DMA controller generates an interrupt when the complete message enters the network. From that point, the Direct Connect hardware will ensure reliable delivery of the message to the destination node.

Message reception follows the same basic steps in reverse. The DMA is kept set up at all times with a place to put a complete message. When a message arrives, an interrupt is generated and the system interprets the message header. When necessary, the system realigns data in the receive buffer.

NX/2 controls the destination of each message by generating a byte of routing information which is interpreted by the Direct Connect hardware. When the cube is partitioned into subcubes, the system software translates subcube node numbers (which are numbered starting at zero for each subcube) into physical node numbers when messages are sent, and translates back when messages arrive.

3. Message Protocols

NX/2 provides flow control and message buffering. This removes much of the burden of deadlock prevention from the user. A common situation arises in some applications when several nodes send messages to a single node. This occurs even in fairly well-synchronized applications, where a node will interchange messages with only one other node after each phase of computation. If, for one reason or another, a node gets behind other nodes in its computations, the other nodes can send it messages which are from phases in its future. In order for the node to proceed, it must be able to receive the message for its next phase even if there are other messages waiting. In addition, these other messages must not be lost, since they will be required eventually.

NX/2 provides two levels of protocol to avoid deadlock in this kind of situation. The lowest level is based on very short messages, and the higher level covers all longer messages. The protocols are also designed to deliver high performance. They rely on the reliable message passing capabilities of the hardware, which has its own even lower level of protocol for establishing a route and providing limited flow control and buffering.

3.1 Short Messages

Messages of 100 bytes or less, including both application messages and operating system control messages, use a one trip protocol. There are a large number of short

message buffers maintained by the operating system. Each node allocates some of these buffers to each other node. When a node has a short message to send to another node and there is a buffer reserved for it, it simply sends the message. If the reserved buffers have been used up, it holds the message until buffers are returned. Nodes keep track of incoming messages and periodically return the buffers to the originating nodes. Often, the number of buffers returned can be piggybacked on a returning message.

The latency between the time a process sends a null (0 byte) message from one node and a process on a neighboring node receives it is about 350us. Non-null messages up to 100 bytes take slightly longer due to the time required to parameter check the buffer pointers in the send and receive calls and the time to copy the data. Control messages sent between nodes from within the operating system take less time.

3.2 Long Messages

Messages longer than 100 bytes use a three-trip protocol. In the first step, the system sends a control message which serves as proxy for the entire message. The proxy can be saved in a short message buffer on the receiving node until there is a place for the whole message. If there is a receive posted for the message, or there are enough free pages of memory to contain the entire message, the system sends back a control message requesting that the rest of the message be sent. It also sets up the DMA controller to place incoming message bytes beyond the size of a short message in the receive buffer. When the sending node receives the request message it immediately sends the entire long message.

Because of the way the DMA is set up, the first 100 bytes of a long message are received into a short message buffer. This ensures that short messages arriving ahead of the long message are treated properly. The rest of the long message is DMA'ed directly into the receive buffer. The system then realigns the data if necessary and copies the first 100 bytes.

This protocol provides flow control and controlled buffering by requiring only a short message buffer on a receiving node regardless of the size of an incoming message. In addition, the practice of sending the entire message at once eliminates a performance ceiling imposed by packetization. This means that message passing bandwidth asymptotically approaches the hardware bandwidth of 2.8 Mbyte/sec as the message size increases. In fact, very large messages do achieve over 2.7 Mbyte/sec rates.

4. Application Interface for Message Passing

Applications access the message passing capabilities of the system through system calls. NX/2 provides nested sets of system calls with a range of complexity and power. At the lowest level, there is a complete set of simple blocking calls and information calls. Another level provides asynchronous calls for overlapping processing with message passing. The final level contains calls for interrupt-driven message passing.

Calls from different levels can be freely mixed, and in fact the higher levels rely on calls in lower levels. For example, a message sent with an asynchronous call can be received with a simple blocking call or an interrupt-driven call. In general, the lower level calls are simpler to use and consume less processor time,

while the higher level calls provide more capability (and therefore more opportunity to utilize the processor efficiently) at the cost of increased processor overhead per call.

At all levels, messages have common characteristics. Messages can be any length, from zero bytes to the limit of physical memory. Message destination is determined by node number and process id. Each message has a type which can be used for identification or to enforce message order.

The *typesel* parameter in receive and probe calls selects messages by type. There are three cases. When the *typesel* parameter is positive, it explicitly selects that unique type. When *typesel* is -1, all types are selected. Otherwise, when the sign bit is set in *typesel*, the other bits form a mask which allows selection of any set of types in the range 0-30.

4.1 Message Passing Models

These characteristics are designed to effectively support program-directed message passing. In this model, message traffic is strictly determined by the sequential execution of program code by using send and receive calls within each node process. NX/2 provides message ordering, typing, flow control, and buffering so that each process, running at its own rate, can receive messages in their expected order. At the same time, it minimizes the possibility of deadlock so that no special attention is required on the part of the application programmer. Experience indicates that this model of programming is easy to learn and use, both for original code and for porting existing sequential applications to concurrent execution. It is an especially good match for data domain decomposition techniques.

The generality of the message passing system calls is sufficient to support other message passing models with reasonable efficiency. Applications can be built from clusters of identical or unique processes, with separate address spaces using the built-in process control or as light weight shared address space processes constructed within a single NX/2 process. These processes can communicate via RPC mechanisms or can use an object oriented, message-driven approach. In many cases it will be necessary to write additional support code at the application level to implement these models, but in all cases the NX/2 message passing calls should provide a convenient base.

4.2 Simple Blocking Calls

The core message passing calls are sufficient to write a complete concurrent application. They send, receive, or wait for a message to arrive, and return when the operation is complete.

```
csend ( type, buf, length, node, pid )
crecv ( typesel, buf, length )
cprobe ( typesel )
```

Simple Message Passing Calls

The *csend()* call assigns a type and destination to a message, then sends it. The call does not return until the message has entered the communications network and the send buffer is no longer needed. Note that the *csend()* call does not necessarily wait for the message to be received at its destination.

The *crecv()* call selects an incoming message by type and receives it into a buffer. It does not return until the message arrives in the buffer.

The *cprobe()* call can be used to wait for a selected message or one of a set of messages to arrive at the node. When *cprobe()* returns, *crecv()* can be used to direct the message into a buffer.

There are four *info* calls which return information about the most recent message received or probed for. They might be used after *cprobe()* to determine the correct buffer to use in the subsequent *crecv()* call.

```
infotype ()
infonode ()
infopid ()
infocount ()
```

Message Information Calls

There are also several calls which allow a process to identify itself and determine its position in the system environment.

```
mynode ()
mypid ()
numnodes ()
```

General Information Calls

4.3 Asynchronous Calls

Simultaneous message passing and computation are possible by using calls which return as soon as the operation is initiated.

```
mid =
    isend (type, buf, length, node, pid)
mid = irecv ( typesel, buf, length)
```

Asynchronous Message Passing Calls

The *isend()* call initiates transmission of a message and immediately returns a message id. The message id must be used to determine when the message enters the communication network and the message buffer is no longer needed. Again, this does not allow the sender to determine when the message is actually received.

The *irecv()* call posts a request to receive a message into the buffer. It also immediately returns a message id which must be used to determine when the message arrives. The *irecv()* call is particularly useful for setting up the receive buffer for an incoming message before the message is required in the application. This is efficient for two reasons. First, when the message arrives the system will take less overhead in determining what to do with it, since it can be received directly into the receive buffer. Second, the application can do useful computation while the message is arriving.

```
msgwait
flag = msgdone (mid)
flag = iprobe (typesel )
```

Status Calls

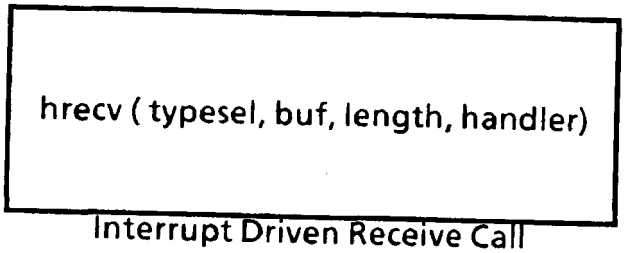
The *msgwait()* and *msgdone()* calls must be used to determine when an asynchronous send or receive operation completes. They both take the message id of the operation as a parameter. *Msgwait()* does not return until the operation completes, while *msgdone()* immediately returns a boolean which is true if the operation is complete. After either of these operations indicate that a receive operation is complete, the *info()* calls from the basic subset can be used to obtain information about the message.

The *iprobe()* call provides a means of determining whether a message is ready to be received. Unlike the *cprobe()* call, the *iprobe()* call does not

wait for the message but immediately returns a boolean which is true if the message has arrived. In that case, the *info* calls can be used to find out about the message.

4.4 Interrupt-driven message passing

When more independence is needed between message passing and processing, message reception can be interrupt-driven.



Interrupt Driven Receive Call

The *hrecv()* call posts a request to receive a message into the buffer and returns immediately. It also sets up a handler procedure which will be called when the receive operation is complete. The handler is called with parameters containing information about the message, so that the *info* calls are not needed.

It is possible to post several *hrecv()* requests simultaneously, with separate handlers or with a common handler. In addition, receive completion interrupts can be temporarily blocked to protect critical sections.

5. Future Direction

As the machine evolves, the operating system will necessarily evolve with it. The next major enhancement to NX/2 will support new I/O hardware which connects disk, tape, and other I/O devices directly to the cube.

We are also investigating several of the new programming models, such as SVM (shared virtual memory) and various object oriented approaches. SVM and some object oriented models rely on RPC (remote procedure call) mechanisms, while other models rely on very simple, high speed, low overhead message passing.

We would like to add support for SVM into the NX/2 kernel, making it an extension of the existing process model. However, for some of the new message passing models the inherent overhead of the flexible message passing model supported by NX/2 is too high, and a different approach is required. For these models, NX/2 can provide a convenient environment for initializing the hardware and loading the new system code, then step out of the way and provide direct access to the message passing hardware. We are looking at ways of allowing low overhead message passing to co-exist with the current model, so that we can continue to use existing tools (such as the debugger) and so that applications written to different programming models can interact.

We are also looking into possibilities for additional support of the existing programming model, such as higher level message passing functions that allow global arithmetic and logical operations to perform at kernel speeds.

**Unix* is a trademark of ATT&T Bell Laboratories.

The Intel iPSC/2® System:

The Concurrent Supercomputer for Production Applications

SUMMARY

The iPSC®/2 concurrent supercomputer delivers a new standard of price performance for scientific and engineering computation: 10 times that of a supercomputer and 100 times that of a supermini. On actual application codes the iPSC/2 system has demonstrated true supercomputer performance (over 100 MFLOPS).

The iPSC/2 system, which began shipping in December 1987, is a powerful applications engine that can offload the compute-intensive portions of large scientific and engineering applications.

At the same time, the iPSC/2 system offers an interactive, multi-user environment via the Sun*-3 workstation. Users retain their familiar applications interfaces but reap the benefits of supercomputer power: large problems solved, with higher-quality results, in dramatically less time.

iPSC/2 SYSTEM OVERVIEW

System: The iPSC/2 system (see figure 1) is a highly parallel, scalable computer system. The system's multiple processing elements, or nodes, are complete, self-contained computers with substantial execution speed and local memory capacity (see table 1). Nodes vary in processing power and memory capacity.

Figure 1. The Intel iPSC®/2 system

The Network: A high-performance network, optimized for message-oriented communication, interconnects the nodes. The iPSC/2 Direct-Connect™ communication network provides a high-speed data pathway between the nodes of the system, and between any node inside the system and the System Resource Manager.

Nodes: The nodes of the iPSC/2 system offer exceptional flexibility

for the system designer or application programmer. The basic iPSC/2 node features a 32-bit Intel 80386 microprocessor and 80387 floating point co-processor, one to sixteen megabytes of memory, and a Direct-Connect network interface. Two options (the iPSC-SX and iPSC-VX) are available to accelerate numeric performance at each node.

Development Environment: The UNIX-based software development environment for the iPSC/2 system supports a multi-person programming team in a familiar workstation setting, while providing a set of state-of-the-art tools for developing and debugging concurrent applications. Known as the Concurrent Workbench™ software, the software includes: C, Fortran, and Common LISP languages; the VAST-2 vectorizer; linker; libraries; utilities; and DECON, the iPSC/2 system's concurrent debugger.

iPSC/2 APPLICATIONS

The iPSC/2 system is a highly cost-effective solution for scientific and engineering applications that demand large scale performance. Among the applications running or being developed for the iPSC/2 system:

Computational Mechanics: Advanced modeling techniques for engineers, such as non-linear structural analysis, 3-dimensional fluid dynamics, and aerodynamics, have taxed the capabilities of conventional supercomputers. These computationally-demanding techniques are perfectly suited to the iPSC/2 system's combination of concurrent and vector computing.

Intel is working with partners in the field of computational mechanics to build a full set of complementary software applications for the mechanical engineer. These include:

• NEKTON*, from Nektonics Incorporated, is a state of the art package for fluid dynamics and heat transfer analysis. NEKTON solves the full incompressible, unsteady

Table 1: iPSC®/2 system specifications

CUBE DIMENSION Number of Nodes		d3 8	d4 16	d5 32	d6 64	d7 128
AGGREGATE MEMORY						
Basic and iPSC-SX systems	1 MByte/node	—	16	32	64	128
	4 MByte/node	—	64	128	256	512
	8 MByte/node	—	128	256	512	1024
	16 MByte/node	—	256	512	1024	—
iPSC-VX system	1 MByte/node—1 MByte VX	16	32	64	128	—
	4 MByte/node—1 MByte VX	40	80	160	320	—
	8 MByte/node—1 MByte VX	72	144	288	576	—
AGGREGATE PERFORMANCE (PEAK)						
MFLOPS—32-bit Precision	iPSC/2 basic system		4.0	8.0	16.0	32.0
	iPSC/2 SX scalar option	—	18.0	35.0	70.0	141.0
	iPSC/2 VX vector option	160.0	320.0	640.0	1280.0	—
MFLOPS—64-bit Precision	iPSC/2 basic system		3.4	6.7	13.0	27.0
	iPSC/2 SX scalar option	—	10.0	20.0	40.0	81.0
	iPSC/2 VX vector option	53.0	106.0	212.0	424.0	—
MIPS	All systems	32	64	128	256	512

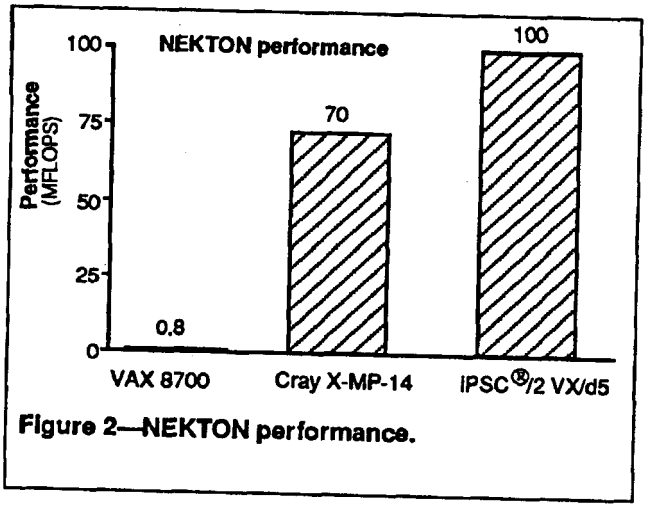


Figure 2—NEKTON performance.

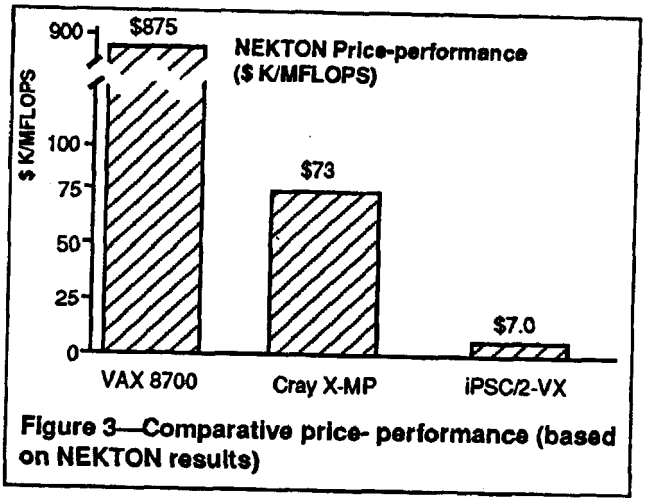


Figure 3—Comparative price- performance (based on NEKTON results)

Navier Stokes and energy equations and is applicable to a wide range of problems in the aerospace, automotive, biomedical, electronics, and process industries. Simulations running on NEKTON on the iPSC/2 system have achieved supercomputer performance (see figure 2), and a 10-fold improvement in price-performance over conventional supercomputers.

• PASSAGE* from Technalysis incorporated is a powerful software package for the analysis of internal fluid flows. PASSAGE is based on an advanced block-structured finite element technique and is being used in automotive and aerospace applications. Early benchmarks suggest that PASSAGE performance, delivered on a 32-node iPSC-SX system, will exceed the capability of a Cray X-MP supercomputer.

• Additional software development is underway in the area of modeling the aerodynamics of high-performance airframe design.

Discrete event simulation: Using simulation tools on the iPSC/2 system, programmers can focus on the task of building very large discrete event simulation models, instead of the details of the system architecture.

• Interwork II*, from Block Island Technologies, provides several useful tools for developing concurrent programs spanning a variety of applications. It offers an extensive set of tools to assist in application development, and its object orientation promotes the construction of modular, reliable programs.

Petroleum Exploration and Production: The iPSC/2 system's combination of vector and concurrent computing provides performance beyond typical supercomputers—for seismic modeling and oil reservoir simulation.

• A state-of-the art seismic processing application is being developed by a software company for the iPSC-SX system. Employing a "model-based" seismic processing technique, the application takes raw seismic data and employs a sophisticated ray tracing method to drive a self-correcting model of sub-surface geology.

Molecular mechanics and structure optimization: Molecular modeling predicts the structure and behavior

of complex chemical formulations by simulating the chemistry and physics of the constituent atoms. Applications development is progressing on techniques ranging from ab initio quantum mechanical techniques to molecular mechanics and dynamics.

- HYPERNEWTON, from Hypercube Incorporated, is a general purpose package for molecular mechanics and dynamics calculations. When development is completed, it can be used for the structural determination of organic molecules and is suitable for protein and nucleic acid engineering applications.

- Additional software under development will predict the 3-dimensional structure of macromolecules. It is based on a proprietary energy minimization technique and is targeted for the iPSC/2 VX architecture.

Electronic Design: Continuing increases in complexity and the competition for faster time to market are driving electronic designers to total computer simulation of systems, chips, and processes. Intel Corporation is targeting the iPSC family for VLSI design applications including process, device, and circuit simulation.

iPSC/2 REFERENCES

The following papers on the iPSC/2 system appear in the Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications:

Arlauskas, Ramune. "iPSC/2 System: A Second Generation Hypercube."

Arshi, Shala; Asbury, Ray; Brandenburg, Joe; and Scott, David ."Application Performance Improvement on the iPSC/2 Computer."

Bain, William L., and Arshi, Shala. "Hypersim: Hypercube Simulator for Parallel System Performance Modeling."

Close, Paul. "The iPSC/2 Node Architecture."

Ertel, Dave. "Environment for Enhancing iPSC/2 Programmer Productivity."

Nugent, Steven F. "iPSC/2 Direct-Connect Technology."

Pan, Wei Min, and Jackson, Victor. "A Concurrent Debugger for iPSC/2 Programmers."

Pierce, Paul. "The NX/2 Operating System."

FOR MORE INFORMATION

To learn more about the iPSC/2 system and applications, or about concurrent computing training classes and seminars, contact:

Intel Scientific Computers
15201 N.W. Greenbrier Parkway
Beaverton, OR 97006
(503) 629-7629