Network-Based Multicomputers: An Emerging Parallel Architecture

H.T. Kung, Robert Sansom, Steven Schlick, Peter Steenkiste, Matthieu Arnould,
Francois J. Bitz, Fred Christianson, Eric C. Cooper, Onat Menzilcioglu, Denise Ombres, Brian Zill

School of Computer Science / Carnegie Mellon University / Pittsburgh, PA 15213

Abstract

Multicomputers built around a general network are now a viable alternative to multicomputers based on a system-specific interconnect because of architectural improvements in two areas. First, the host-network interface overhead can be minimized by reducing copy operations and host interrupts. Second, the network can provide high bandwidth and low latency by using high-speed crossbar switches and efficient protocol implementations. While still enjoying the flexibility of general networks, the resulting network-based multicomputers achieve high performance for typical multicomputer applications that use system-specific interconnects. We have developed a network-based multicomputer called Nectar *that supports these claims.*

1 Introduction

Current commercial parallel machines cover a wide spectrum of architectures: shared-memory parallel computers such as the Alliant, Encore, Sequent, and CRAY Y-MP; and distributed-memory computers including MIMD machines such as the Transputer [15], iWarp [5, 6], and various hypercube-like systems [3, 17], and SIMD machines such as the Connection Machine [26], DAP, and MasPar. Like SIMD machines, distributed memory MIMD computers, or *multicomputers*, are inherently scalable. Multicomputers however can handle a larger set of applications than SIMD machines because they allow different programs to run on different processors. Multicomputers with over 1,000 processors have been used successfully in some application areas [14].

Multicomputers are traditionally built by using system-specific interconnections to link a set of dedicated processors. Examples of these traditional multicomputers are any of the high-performance hypercube systems such as the iPSC-2 and N-Cube machines. Like a proprietary internal bus in a conventional machine, the interconnect is intended to connect to a small set of specially-designed processor boards, and is optimized to do so. System-specific interconnects are used instead of general networks, such Ethernet or FDDI, mainly for performance reasons. However, since they are system-specific, these interconnects do not have the flexibility of general networks: for example, they cannot be connected to many types of existing hosts.

This paper argues that it is feasible to build high-performance *network-based multicomputers* that use general networks instead of system-specific interconnects. Such a multicomputer is able to use *existing* hosts, including workstations and special-purpose processors, as its processors. While enjoying a high degree of flexibility, the underlying network can have performance comparable to that of a dedicated interconnect. This is true both for large messages, where bandwidth is important, and for small messages, where software overhead is typically the limiting factor. As a result, the types of applications that run on existing multicomputers also run efficiently on network-based multicomputers. At the same time, network-based multicomputers are able to take advantage of rapid advances in network and processor technology.

The *Nectar* project is one of the first attempts to build a high-performance network-based multicomputer. Nectar is composed of a high-bandwidth crosspoint network and dedicated network coprocessors. A prototype system using 100 Mb/s (megabits per second) links has been operational since early 1989. The system has currently 26 hosts, including a connection to a CRAY Y-MP via a 26 kilometer single-mode fiber link. The Nectar prototype has been used as a vehicle to study architectural issues in network-based multicomputers and high-speed networks. This paper is based on insights and experiences gained from the Nectar prototype.

To further pursue our ideas, we are collaborating with an industrial partner (Network Systems Corporation) to develop a *gigabit Nectar* system capable of sustaining gigabit per second end-to-end communication. This new system will support the 800 Mb/s HIPPI (High-Performance Parallel Interface) ANSI standard, and will also have a SONET/ATM

This research was sponsored by the Defense Advanced Research Projects Agency (DOD) under contract number MDA972-90-C-0035, in part by the National Science Foundation and the Defense Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives.

interface supported by phone carriers. The gigabit Nectar system is one of the five testbeds in a US national effort to develop gigabit per second wide-area networks [24].

This paper describes architectural features that are desirable for general-purpose networks if they are to be used as interconnects for high-performance multicomputers. In Section 2 we summarize the advantages and interconnection requirements of network-based multicomputers and in Section 3 we give an overview of the prototype Nectar system. We then look at design tradeoffs of critical network components: the host-network interface (Section 4) and the network interconnect (Section 5). We summarize our results in Section 6.

2 Network-Based Multicomputers

We first summarize the advantages of network-based multicomputers and then describe the architectural requirements for their interconnect.

2.1 Network-Based Multicomputer Advantages

Network-based multicomputers offer substantial advantages over traditional multicomputers that use dedicated interconnects:

1. *Supports heterogeneous architectures.* A network-based multicomputer can incorporate nodes specially selected to suit a given application. Instead of having computations with different characteristics use a single architecture as in conventional computer systems, network-based multicomputers support a new computation paradigm that matches architectures to different computational needs.

2. *Use of existing architectures.* By incorporating *existing* systems as hosts, a network-based multicomputer can take advantage of rapid improvements of commercially available computers. In addition, it can reuse existing systems software and applications.

   In fact, a network-based multicomputer provides a graceful environment for moving applications to new architectures such as special-purpose, parallel systems. In the beginning, an application can execute part of the computation on these new systems while using more conventional systems to run the rest of the application. The application can increase its use of the new systems as more software and application code for the new systems is developed.

3. *Availability of large data memory.* In a network-based multicomputer, applications can use the aggregate of the memory available in all the nodes. Since each node can have a sizable memory, the amount of memory is potentially huge, and it becomes possible to solve very large problems using relatively modest systems such as workstations.

4. *High-speed I/O.* The underlying high-speed network is inherently suited to support high-speed I/O to devices such as displays, sensors, file systems, mass stores, and interfaces to other networks. For example, via such a network, disk arrays [18, 21] can deliver very high data transfer rates to applications. Thus, because of the combination of their I/O capability and their ability to incorporate powerful computing nodes, network-based multicomputers represent a balanced architectural approach capable of speeding up both computation and I/O.

2.2 Multicomputer Interconnect Requirements

The requirements for a multicomputer network are a combination of the requirements of general-purpose networks and dedicated interconnects. In the first place they must have the high *performance* of the dedicated interconnects found in traditional multicomputers so that they can support multicomputer applications efficiently, but at the same time, since they have to operate in the same environment as a general-purpose network, multicomputer networks must have the same *flexibility* and *reliability* as general-purpose networks.

Performance has both throughput and latency requirements: for large messages, the network bandwidth determines how quickly a message can be delivered; however, for small messages, having a low overhead on the sender and receiver side is more important. In the case of network-based multicomputers, the network efficiency must increase in proportion to the computation rate of the hosts on the network. For today's traditional multicomputers, the bandwidth of each interconnection link can be as high as several 100 Mb/s [5] and the communication latency between processes on two processors can be as low as 50 to 500 microseconds [4]. These numbers set performance goals for network-based multicomputers, and they have an impact on both the design of the network and the host-network interface.

In terms of flexibility, the network must be general-purpose enough to allow the attachment of many types of computers. Moreover, the nodes of network-based multicomputers are typically distributed over a building or campus so regular interconnect architectures such as a torus or hypercube are not practical. Although regular interconnects are attractive when mapping regular algorithms on homogeneous multicomputers, they are not flexible enough to allow the matching of network bandwidth to the requirements of heterogeneous hosts, or to handle the adding and removing of nodes while the network is operating.

As in any general-purpose LAN, the underlying network of a network-based multicomputer needs to cope with data errors and network failures, since the same physical media are

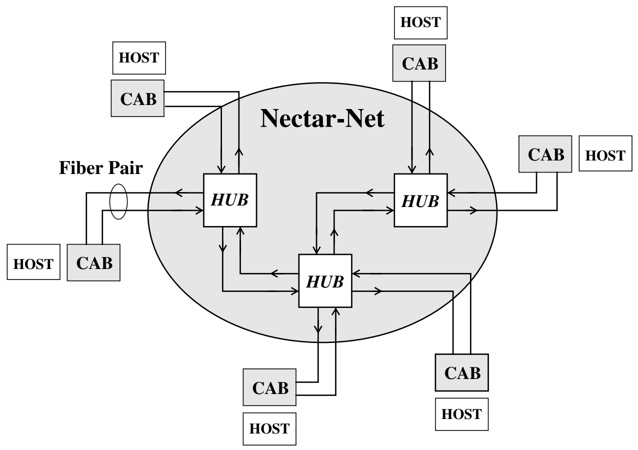


Figure 1: Nectar system

used in both cases. This is in contrast with traditional multicomputers, which typically assume that the interconnect is reliable. One of the challenges of building multicomputers around general networks is to make sure that the techniques employed to insure reliability do not hamper the system performance.

In the rest of the paper we describe methods of achieving these requirements. We start by describing the Nectar system, our first system experiment in this area.

3 Nectar System

To demonstrate the feasibility of network-based multicomputers, we started developing the Nectar system [2] in 1987. Nectar is a high-bandwidth, low-latency computer network for connecting high-performance hosts. Hosts are attached using Communication Acceleration Boards (CABs). The Nectar network consists of fiber-optic links and crossbar switches (HUBs). The HUBs are controlled by the CABs using a command set that supports circuit switching, packet switching, multi-hop routing, and multicast communication. Figure 1 gives an overview of the Nectar system.

3.1 Nectar Prototype

We have built a 26-host *Nectar prototype* system to support early system software and applications development. The prototype uses 100 Mb/s fiber links and 16×16 HUBs. The CAB is implemented as a separate board on the host VMEbus.

The CAB is connected to the network via a fiber port that supports data transmission rates up to 100 Mb/s in each direction. The fiber port contains the optoelectronics interfaces to the two fiber lines and FIFOs to buffer data transferred over the fibers. Each CAB has 1 megabyte of data memory, 512 kilobytes of program memory, and a 16.5 MHz SPARC processor. The CAB also has a DMA controller to provide high speed transfers between the fiber port and the data memory, and between the data memory and the VMEbus interface.

3.2 Nectar Systems Software

The Nectar system software consists of a CAB runtime system and libraries on the host that exchange messages with the CAB on behalf of the application. The CAB runtime system manages hardware devices such as timers and DMA controllers, supports multiprogramming (the threads package), and manages data buffers (the mailbox module). The threads package, derived from the Mach C Threads package [9], supports lightweight threads in a single address space. Threads provide a low cost, flexible method of sharing the CAB CPU between concurrent activities, which is important for communication protocol implementation. Mailboxes provide flexible and efficient management of buffer space in the CAB memory and form the endpoints of communication between processes on hosts or CABs.

The streamlined structure of the CAB software has made it possible for Nectar to achieve low communication latency. For the existing Nectar prototype, the latency to establish a connection through a single HUB is under 1 microsecond. The latency is under 100 microseconds for a message sent between processes on two CABs, and about 200 microseconds between processes residing in two workstation hosts. The above figures do not include the fiber transmission latency of approximately 5 microseconds per kilometer. These performance results are similar to those of traditional multicomputers.

The CAB runtime system currently supports several transport protocols with different reliability/overhead tradeoffs [10]. They include the standard TCP/IP protocol suite besides a number of Nectar-specific protocols. For TCP, when TCP checksums are not computed, the throughput between two CABs is over 80 Mb/s for 8 kilobyte packets. When checksums are computed, TCP throughput drops to about 30 Mb/s. This indicates that hardware support for checksum calculation can significantly improve performance, at least for this type of system.

3.3 Nectar Applications

The network-based multicomputer architecture has made it possible to parallelize a new class of large applications [19]. These applications were previously either too large or too complex to be implemented on parallel systems. We have successfully ported several such applications onto the Nectar prototype system. Examples are COSMOS [7], a switch-level circuit simulator; NOODLES [8], a solid-modeling package; and a simulation of air pollution in the Los Angeles area.

Because Nectar uses existing general-purpose computers as hosts, applications can make direct use of code that has previously been developed for these computers, and as a result, the Nectar implementation of the above applications took relatively little effort in spite of their relatively high complexity. In the distributed versions of both COSMOS and Noodles, for example, each node executes a sequential version of the program that was modified to do only part of the computation.

The COSMOS simulator was ported to Nectar by partitioning the circuit across the Nectar nodes. This makes it possible to simulate very large circuits, that cannot be handled on a single node, and it illustrates the benefit of being able to use the aggregate memory of a number of systems for a single application. Other applications, such as a chemical flow sheet application, were able to use a group of workstations plus a Warp systolic array.

In addition to the use of existing code, the implementation of applications on Nectar has emphasized the use of general-purpose supports for large-grain parallelization. This development approach complements existing fine-grain efforts in parallelizing inner-most loops of computations. The combined capability should significantly increase the applicability of parallel processing.

4 Host-Network Interface

The critical factor in parallelizing applications on a multicomputer is how quickly tasks on different hosts can communicate. The latency is often determined by software overhead on the sending and receiving hosts, so reducing this overhead is a primary goal in the design of a host-network interface. In Nectar we decreased message latency by reducing the number of data copies for each message (each copy adds significant latency because main memory bandwidth is limited on most hosts), and by offloading protocol processing to an outboard processor so that the number the number of host interrupts is reduced (each host interrupt adds 10-20 microseconds to latency [1]).

We first describe three design alternatives for the host-network interface, and examine how different software implementations can utilize these designs. We then discuss specific hardware and software issues for building interfaces for workstations, based on our experience with Nectar.

4.1 Host-Network Interface Design

The three components that play a role in the host-network interface are the host CPU, main memory, and network interface. Figure 2 shows three ways in which data can flow between these components when sending messages. The grey arrows indicate the building of the message by the application; the black arrows are copy operations performed by the system.

The architecture depicted in Figure 2(a) is the network interface found in many computer systems, including most workstations. When a system call is made to write data to the network, the host operating system copies the data from user space to system buffers. Packets are sent to the network by providing a list of descriptors to the network controller, which uses DMA to transfer the data to the network. The main disadvantage of this design is that three bus transfers are required for every word sent and received. However, this is not really a problem if the speed of the network medium is sufficiently slow compared to the memory bandwidth, as is the case for current workstations connected to an Ethernet.

The communication activity relative to the processing activity is much higher on multicomputers than on traditional general-purpose networks, and as a result, the network speed of multicomputers has to be a significant fraction of the processor and memory bandwidth of the computer to avoid that the network becomes a bottleneck. Existing workstations connected to a high-speed network (100 Mb/s or higher bandwidth) are an example. In these systems, the memory bus will be a bottleneck if the architecture of Figure 2(a) is used for the network interface. The performance can be improved by reducing the number of data copies done over the memory bus by using external memory in the network interface. Figures 2(b) and 2(c) show two alternative ways of utilizing external memory.

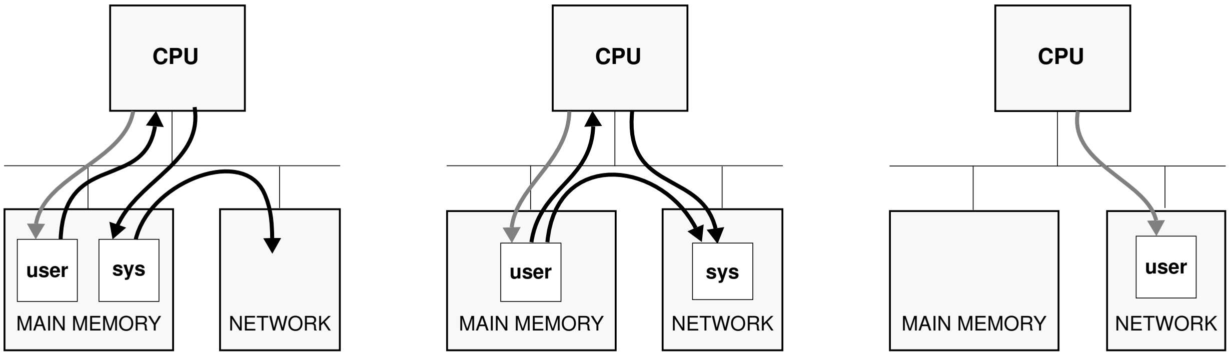
Figure 2(b) depicts an alternative in which the system buffers have been moved from host memory to external memory on the network interface. When data is sent or received, the data is copied between the user buffer in main memory and the system buffers in the network interface. The copying requires two bus transfers per word if it is done by the CPU, and one bus transfer per word if it is done by a DMA controller.

The approach used in Nectar is shown in Figure 2(c). With this approach the user buffers are located on the network interface (the CAB), and the data does not have to be copied: data packets are formed and consumed in place by the user process. As a result, communication latency is minimized and main memory bandwidth is conserved.

4.2 Network Interface Software

The design alternatives shown in Figure 2 are linked to the ways in which applications send and receive messages. With the Unix socket interface [20] users specify messages with a pointer-length pair. The semantics of the Unix socket read call is that the call returns when the message is available in the specified area. The socket write call returns when the message can be overwritten. The semantics of both calls requires that the data is logically copied as part of the call. This requirement is naturally met by the standard host interface implementation of Figure 2(a), although the design of Figure 2(b) can also be used in the socket model.

To support the host interface of Figure 2(c), the Nectar interface implements "buffered" send and receive primitives [23] using the mailboxes mentioned in Section 3.2.



(a) (b) (c)

Figure 2: Design alternatives for the host-network interface

With a buffered send, the application builds its message in a message buffer that was previously obtained from the system. As part of the send operation, the application gives up the right to access the message buffer, so the system can free it automatically when the data has been sent and appropriately acknowledged. Similarly for a receive, the system returns a pointer to a message buffer to the user process. The application has to return the message buffer to the system after the message has been consumed.

The advantage of buffered sends and receives over socket-like primitives is that data no longer has to be copied as part of the send and receive calls. This gives the implementation more freedom in implementing the communication interface, and can result in a lower overhead to the application and lower message latency.

Our experience with the Nectar system indicates that buffered primitives are faster for large messages, but that the immediate (socket-like) primitives are faster for short messages. The reason is that for immediate sends and receives of short messages, the data can be included with the request that is exchanged between the host and the CAB. For short messages, the extra complexity of the buffer management for buffered primitives is more expensive than the cost of simply copying the data. In the Nectar prototype, the buffered primitives are more efficient for messages larger than about 50 words.

4.3 Design Choices

We review the major design choices that were made for the prototype Nectar system and draw some general conclusions based on our experience with the prototype.

4.3.1 Shared Memory

One problem with the shared memory interface in the Nectar prototype is the relatively high latency of CAB memory accesses from the host. The access time to CAB memory is approximately 2 to 3 times greater than to main memory, depending on the particular host. A large part of this latency is due to the asynchronous VME bus that requires both the host and CAB to synchronize on every transfer. More recently-developed, high-speed synchronous busses [12, 25] couple the host more tightly to the I/O bus, thus reducing the latency of accesses across the I/O bus.

Maintaining consistency between the host cache and the CAB memory is another problem that must be addressed. The current solution is to mark all buffers as uncacheable, which increases the latency for accesses to messages that are handled using buffered sends and receives. If the CAB memory were cached, it would be necessary to invalidate cache lines when new data arrives on the CAB.

Immediate sends and receives are implemented by having the system software copy the data between the user buffer in main memory and the CAB. Alternatively, the network interface can implement block transfers between user buffers in main memory and the network interface using DMA. Compared with copying using the CPU, using DMA incurs an overhead to pin the affected user virtual memory pages and to invalidate cache lines. Whether CPU copies or DMA transfers are more efficient depends on the relative costs for the particular system. For large transfers these costs can often be amortized, while for small transfers it is typically better to have the CPU read and write the user buffers directly, avoiding the overheads of DMA.

4.3.2 Checksum Hardware

Most software implementations calculate the transport-level checksum in a separate pass over the data. As a result, one memory read is added for every word sent or received. This extra memory access can be avoided by calculating the checksum while the data is copied, either using the CPU or special hardware in the DMA unit. The design shown in Figure 2(b), together with copying and checksumming by the CPU, corresponds to Jacobson's proposed "WITLESS" interface [16].

Since the cost of implementing checksum hardware is typically small, it is attractive to calculate the transport-level checksum in hardware. In particular, when the user buffers reside on the network interface, or when DMA is used to transfer data between host memory and the network interface, calculating the checksum on the network interface means that the host system software does not have to touch the user data at all.

4.3.3 Programmable CPU

The programmable CPU on the CAB is a key feature of the Nectar prototype. The motivations for building the CAB around a general-purpose CPU with a flexible runtime system, were that we wanted to experiment with different protocol implementations and open up the CAB to applications. For a prototype system, this flexibility is more important than the increase in performance that could be achieved with a complete hardware implementation.

In the Nectar prototype, protocol processing can be completely offloaded to the CAB. The main benefit of doing so is that the interaction between the host and the network interface is reduced to a single operation in which the application presents or accepts the data. The actual protocol processing overhead is small compared with the OS overhead cost of handling interrupts and system calls. A comparison of the host-host performance of the Nectar-native reliable message protocol (RMP) and the standard TCP/IP protocol shows that throughput is similar, although RMP has a smaller latency [10].

Applications have access to the CAB so that they can exploit the low CAB to CAB latency. How useful this capability is largely depends on whether restructuring application software significantly improves performance. An example in which this is the case is the dynamic load balancing performed by the CAB CPU in applications such as NOODLES (see Section 3.3). The centralized load balancer is placed on a CAB, allowing it to repond to about 10,000 requests for tasks per second.

4.3.4 Inter-Device DMA

The ability to transfer data directly from the CAB to another device on the VME bus has been very useful in the prototype Nectar. Data can be sent to or received from devices such as frame buffers without going through host memory. Direct DMA both reduces latency to the device and conserves host memory bandwidth. The ability to provide this feature depends on the specification of the target bus. Some newer synchronous busses (such as TurboChannel) do not permit inter-device transfers, the justification being that it complicates the system design [12].

4.3.5 Other Host Architectures

Multicomputers based on general-purpose networks have the advantage that they can include a variety of hosts. Some of these hosts place special requirements on the network interface. We look at the communication needs for some interesting classes of hosts: special-purpose supercomputers (such as iWarp and Connection Machine); general-purpose supercomputers (such as Crays and IBM mainframes); and "dumb" devices (such as disk farms and frame buffers).

Special-purpose supercomputers currently often lack adequate external I/O bandwidth. Although the internal aggregate memory and interprocessor bandwidths of these machines are very high, the bandwidth to external networks and the amount of network buffer space are often small. In addition, such machines are often not suited to tasks such as calculating checksums. As a result, it is desirable that the network interface of special-purpose supercomputer provides functionality similar to that provided by our CAB, such as a large buffer area, checksumming hardware, and switch control and datalink hardware.

General-purpose supercomputers are often capable of calculating packet checksums at near-network bandwidth rates using pipelined vector units. In addition, such machines have paths into large central memories with rates greater than a gigabit per second. Thus for general-purpose supercomputers it is less desirable to provide external buffering and checksum hardware.

For "dumb" devices, it is highly appropriate to have a network interface with a general-purpose processor. Such an interface can provide sufficient intelligence to allow a dumb device to be connected directly to the network.

5 Interconnect Architecture

The other crucial component of a network-based multicomputer is the interconnect or network itself. This section describes alternatives for the interconnect and justifies how a general network, as is used in Nectar, is suitable for a multicomputer.

5.1 Interconnect Design

The requirements for an interconnect for a network-based multicomputer are: high throughput (100-800 Mb/s available to each host); low latency (100-500 microseconds between hosts); good scalability (10-1000 attached computers); support for high-bandwidth multicast communication; and robustness so that recovery from network failures is simple. Some of these requirements are similar to those for general-purpose networks, while others, such as the performance goals, are much stronger.

General-purpose LANs based on a shared medium, such as Ethernet, token ring, or FDDI, do not meet the bandwidth and latency requirements of a multicomputer. As more computers are attached to the medium, the bandwidth available to each system decreases and the communication latency increases. On the other hand, LANs based on high-speed switches (such as Autonet [22], HPC [13] and Nectar) do meet the performance requirements of a multicomputers. In such LANs, each attached computer has an exclusive link to a local switch and can communicate with other computers on the same switch at the full bandwidth of the link. The aggregate bandwidth of the network is very high.

During the design of Nectar, we concentrated on the network features that are important for multicomputer usage. For example, we concentrated on minimizing the overhead on the network interface and during the design of the switch, we concentrated on providing a low connection setup time and hardware multicast (see Section 5.3.3), since these features are important for multicomputer applications. Some features that would be needed to operate a large network in a general-purpose environment were not included. For example, we do not attempt to provide automatic network reconfiguration, since the configuration of a Nectar system is relatively stable and changes can be made manually. These features would result in additional hardware (including switch control processors) and software to manage the network. Autonet on the other hand explored the problems involved in managing large, general, switch-based networks. It presents exactly the same interface as an Ethernet to the workstations attached to it and it supports automatic network reconfiguration when hosts or switches are added to or removed from the network.

In the rest of this section, we briefly describe the Nectar network and then discuss specific issues in the design of a network for a multicomputer, including flow control, routing, multicast, and robustness.

5.2 Nectar Network Overview

As described earlier in Section 3, the Nectar network is built around HUBs, which are 16×16 crossbar switches. A HUB implements a command set that allows source CABs to open and close connections through the switch. These commands can be used by the CABs to implement both packet switching, in which case the maximum packet size is determined by the size of the input FIFO on each port of the switch, and circuit switching, in which case the maximum amount of data that can be sent is arbitrarily large.

In the prototype HUB, the latency to set up a connection and transfer the first byte of a packet through a single HUB is ten cycles (700 nanoseconds). Once a connection has been established, the latency to transfer a byte is five cycles (350 nanoseconds), but the transfer of multiple bytes is pipelined to match the 100 Mb/s peak bandwidth of each fiber link. Furthermore, the HUB controller can set up a new connection every 70 nanosecond cycle.

Because of the low switching and transfer latency of a single HUB, the additional latency in a multi-HUB system is not significant. Thus it is practical to build multicomputers consisting of small numbers of these HUBs and up to 100 hosts. Although it is possible to build larger multicomputers using the current HUB, the network management and configuration problems become greater with large numbers of HUBs (see below). A larger HUB crossbar switch (for example, 32×32 or 64×64) would reduce these problems and allow networks of hundreds of hosts to be built and managed.

5.3 Design choices

We review the major design choices that were made for the prototype Nectar system and draw some general conclusions based on our experience with the prototype.

5.3.1 Flow Control

The network must provide flow control hardware to ensure that input buffers at the switches and hosts do not overflow. Lost packets have to be retransmitted by (software on) the sender and have a negative impact on both throughput and latency. Because of the strict performance requirements for multicomputers, avoiding lost packets is even more important for multicomputer networks than for general-purpose networks.

Flow control in Nectar is performed by hardware on the CABs and HUBs under the control of CAB software. The source CAB tests the status of the input buffer on its local HUB before sending a packet. It can check the status of buffers on intermediary HUBs and the destination CAB by using HUB commands provided for this purpose. These commands allow the next packet to start flowing when there is room in the buffer at the other end of a link.

The acknowledgments that tell a CAB or a HUB that there is space in the next buffer are generated by hardware on the next HUB or by software on a destination CAB. On the HUB, the acknowledgment is generated when a packet that is currently stored in the input buffer starts flowing out of the buffer. On the destination CAB, the acknowledgment is generated by the datalink software because only the software can know when there will be space in the input buffer. Typically the datalink software will send the acknowledgment when it has started the DMA to transfer the packet in the buffer to local memory.

Implementing flow control in software has a small overhead cost: it reduces the datalink throughput by less than 10% for 1 kilobyte packets. Nectar implements the flow control in software for flexibility reasons: we want to try out and compare different strategies. Providing full hardware flow control between HUBs and CABs would reduce the overhead.

5.3.2 Routing

For a network the size of Nectar, the routing need not be dynamic or adaptive. Deterministic routing is sufficient, and is in fact preferable since it avoids deadlocks. Deterministic routing can be implemented using source routing. Source routing has the advantage of making the switches very simple as they need no control software or hardware for routing. Source routing does place a small burden on the CAB datalink software to determine routes to remote CABs and to supply the commands needed to open inter-HUB links. A CAB establishes a routing table as part of its initialization by querying the local HUB to determine to which other HUBs it is connected, and doing so recursively on these other HUBs. As is the case with flow control, implementing routing in software adds a small performance penalty: it costs about 1-2% in performance for 1 kilobyte packets.

Nectar uses deterministic source routing because it allows the switches to be very simple. However, for larger networks, source routing has a problem: for packets that must traverse multiple HUBs, the probability of making a connection decreases rapidly with the number of hops. As a result, this approach is only practical for a small number of HUBs. This problem can be fixed in a number of ways. First, dynamic or adaptive routing can be used. This can be done via source routing or by using more intelligent switches that can try a number of alternate routes based on the destination of the packet. Alternatively, routers or bridges can be placed between subnetworks consisting of a small number of HUBs to buffer data that has to travel across too many HUBs. In practice, we expect that a combination of both approaches will be needed for large networks.

5.3.3 Multicast Support

High-bandwidth multicast is more important for multicomputers than for general-purpose LANs because multicomputer applications make significant use of multicast for distributing large amounts of data [19]. Two examples of multicast usage are the initialization of data structures and the distribution of data updates and commands by a master to slave processes. Thus it is desirable that the interconnect provides support for multicast as part of its routing support. Strangely enough, most system-specific interconnects do not provide support for multicast even though it would be useful for applications.

The multicast used by multicomputer applications is different from the multicast or broadcast typically provided in general-purpose networks. The multicast used for network management in general-purpose networks is typically unreliable (i.e. there is no guarantee that all members will receive the message) and the members are not known to the sender. This form of multicast is typically used to retrieve information from (replicated) servers whose location is not known.

The multicast implemented on Nectar is reliable, and the multicast group is created by the sending application. It is implemented by allowing a CAB to have multiple open outgoing connections within one switch. Thus a CAB can set up a multicast connection and send the same data packet to multiple destination CABs.

The source routing used in Nectar makes multicast simple as each host can independently establish its own multicast groups. In networks with switch-based routing, routing information for multicast connections must be entered into the routing tables in the switches. The processor managing the routing table in a switch must manage the multicast groups and inform the hosts attached to the switch about the network addresses that corresponds to particular multicast groups. This adds complexity to the switch.

In either scheme, there must also exist higher-level protocols that manage the membership of multicast groups [11]. These protocols allows hosts to use the same multicast group address (e.g. IP address) to refer to the same set of hosts.

5.3.4 Robustness

Robustness is as important for multicomputer networks as it is for general-purpose LANs. Although the error rates in modern fiber-optic networks are relatively low, is still necessary to implement end-to-end checksums and reliable protocols above the physical layer. The network must be able to recover from errors quickly, otherwise the performance of the multicomputer will degrade rapidly. In contrast, traditional multicomputers typically rely on parity or error correcting codes, since their interconnects have extremely low error rates. If an uncorrectable bit error is detected, the whole multicomputer fails.

End-to-end protocols can recover from network errors that corrupt user data, but dealing with errors that affect control information is more difficult. Network failures in this category include events such as lost flow control acknowledgements or corrupted HUB commands. In the prototype Nectar system, it is the responsibility of the CABs to deal with this type of errors. A source CAB attempts to recover from a failure by resetting the input buffers at each of the HUBs along the route to the current destination CAB. This recovery attempt typically results only in a partial recovery: although the source CAB will have ensured that it has left no partially completed connections or packets in network buffers, it will not have been able to reset the flow control state information. If the loss of a flow control acknowledgement was the cause of the failure, then recovery will not be complete until the destination CAB notices that it has not received a packet or the rest of packet, and sends out an extra flow control acknowledgement.

In general, requiring the network interfaces to perform error detection and recovery is difficult because they may not be able to gather complete information about the state of the network and therefore cannot recover correctly from network failures. For networks of switched media, error detection and recovery should be supported within the switches to make the network robust. The switches should close connections and reset flow control information after some maximum time has been exceeded. By having each switch reset its local state, the failure modes in the whole network are simplified and error recovery is automatic.

6 Summary and Concluding Remarks

Network-based multicomputers represent a new way of constructing parallel systems. In this approach, a parallel system can incorporate existing hosts as its processors. Some of the advantages are that one can select processing nodes that match the requirements of the application, and that existing system and application software can be reused (Section 2.1).

For a network-based multicomputer to deliver high-performance, both its network must meet the following requirements: the host-network interface must have low overhead, and the network must have much lower communication latency and more predictable communication bandwidth than general-purpose LANs. Moreover, the network needs to have much greater flexibility and reliability than the system-specific interconnect of a traditional multicomputer.

As described in this paper, we have found the Nectar prototype to be a useful vehicle for learning about building network-based multicomputers. To achieve high bandwidth, it is important to minimize the number of copy operations in the host-network interface. For short packets however, the cost of copying the data is negligible, and minimizing the interaction between the host and the network interface is more important. Regarding the network interconnect, the Nectar project concentrated on the network aspects that are important for multicomputers. An interconnect based on crossbar switches proves to be attractive, since it can provide a high bandwidth in a cost effective way. Unlike traditional broadcast based network, switch-based networks can deliver increased bandwidth as nodes are added, and unlike the regular interconnects used in traditional multicomputers, they can be configured to meet the needs of a continuously changing set of heterogeneous systems. Hardware support for reliable multicast to a specific subset of nodes, as opposed to the unreliable broadcast typically supported on general networks, proved to be a useful feature for a lot of multicomputer applications.

Our experience with Nectar shows that network-based multicomputers can achieve performance comparable to traditional multicomputers based on dedicated interconnects. In this paper, we have described architectures capable of achieving this goal for workstation hosts, and we have also shown that the systems software can take advantage of the architectural features. For these reasons, we believe that network-based multicomputers are a viable architecture for parallel processing.

References

[1] Thomas E. Anderson, Henry M. Levy, Brian N. Bershad, and Edward D. Lazowska. The interaction of architecture and operating system design. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 108–121. ACM/IEEE, April 1991.

[2] Emmanuel A. Arnould, François J. Bitz, Eric C. Cooper, H. T. Kung, Robert D. Sansom, and Peter A. Steenkiste. The design of Nectar: A network backplane for heterogeneous multicomputers. In *Proceedings of Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, pages 205–216. ACM/IEEE, April 1989.

[3] William C. Athas and Charles L. Seitz. Multicomputers: Message-passing concurrent computers. *Computer*, 21(8):9–24, August 1988.

[4] Shahid Bokhari. Communication overhead on the Intel iPSC-860 Hypercube. ICASE Interim Report 10, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23665, May 1990.

[5] Shekhar Borkar, Robert Cohn, George Cox, Sha Gleason, Thomas Gross, H. T. Kung, Monica Lam, Brian Moore, Craig Peterson, John Pieper, Linda Rankin, P. S. Tseng, Jim Sutton, John Urbanski, and Jon Webb. iWarp: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing '88*, pages 330–339, Orlando, Florida, November 1988. IEEE Computer Society and ACM SIGARCH.

[6] Shekhar Borkar, Robert Cohn, George Cox, Thomas Gross, H. T. Kung, Monica Lam, Margie Levine, Brian Moore, Wire Moore, Craig Peterson, Jim Susman, Jim Sutton, John Urbanski, and Jon Webb. Integrating systolic and memory communication in iWarp. In *Conference Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 70–81, May 1990.

[7] Randy E. Bryant, Derek Beatty, Karl Brace, Kyeongsoon Cho, and Thomas Sheffler. COSMOS: A compiled simulator for mos circuits. In *Proceedings of the 24th Design Automation Conference*, pages 9–16. ACM/IEEE, June 1987.

[8] Young Choi. *Vertex-based Boundary Representation of Non-Manifold Geometric Models*. PhD thesis, Carnegie Mellon University, 1989.

[9] Eric C. Cooper and Richard P. Draves. C threads. Technical Report CMU-CS-88-154, Carnegie-Mellon University, Computer Science Department, June 1988.

[10] Eric C. Cooper, Peter A. Steenkiste, Robert D. Sansom, and Brian D. Zill. Protocol implementation on the Nectar communication processor. In *Proceedings of the SIGCOMM '90 Symposium on Communications Architectures and Protocols*, pages 135–143, Philadelphia, September 1990. ACM.

[11] S. Deering. Host extensions for ip multicasting. RFC 1112, Stanford University, August 1989.

[12] Digital Equipment Corporation. *TURBOchannel Overview*, April 1990.

[13] R. D. Gaglianello, B. S. Robinson, T. L. Lindstrom, and E. E. Sampieri. HPC/VORX: A local area multicomputer system. In *Proceedings of the 9th International Conference on Distributed Computing Systems*. IEEE Computer Society, June 1989.

[14] John L. Gustafson, Gary R. Montry, and Robert E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4):609–638, July 1988.

[15] M. Homewood, D. May, D. Shepherd, and R. Shepherd. The IMS T800 Transputer. *IEEE Micro*, 7(5):10–26, October 1987.

[16] Van Jacobson. Efficient protocol implementation. ACM '90 SIGCOMM tutorial, September 1990.

[17] K. Kaneko, M. Nakajima, Y. Nakakura, J. Nishikawa, I. Okabayashi, and H. Kadota. Processing element design for a parallel computer. *IEEE Micro*, 10(2):26–38, April 1990.

[18] Randy H. Katz, John K. Ousterhout, David A. Patterson, Peter Chen, Ann Chervenak, Rich Drewes, Garth Gibson, Ed Lee, Ken Lutz, Ethan Miller, and Mendel Rosenblum. A project on high performance I/O subsystems. *ACM Computer Architecture News*, 17(5):24–31, 1989.

[19] H. T. Kung, Peter A. Steenkiste, Marco Gubitoso, and Manpreet Khaira. Parallelizing a new class of large applications over high-speed networks. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM SIGPLAN, April 1991.

[20] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[21] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 109–116, June 1988.

[22] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. Research Report 59, DEC Systems Research Center, April 1990.

[23] Peter Steenkiste. A symmetrical communication interface for distributed-memory computers. In *Proceedings of the Sixth Distributed Memory Computing Conference*, Portland, April 1991. IEEE.

[24] G. Stix. Gigabit connection. *Scientific American*, 263(4):118–120, October 1990.

[25] Sun Microsystems, Inc. *Sbus Specification A.1*, January 1990.

[26] Lewis W. Tucker and George G. Robertson. Architecture and applications of the Connection Machine. *Computer Magazine*, 21(8):26–38, August 1988.