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

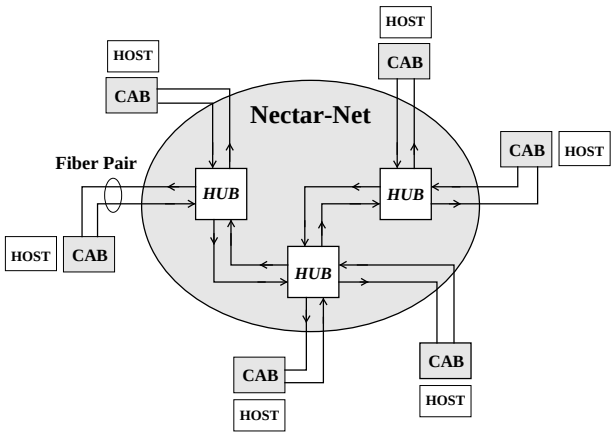

Figure 1: Nectar system

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 $\times$ 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.

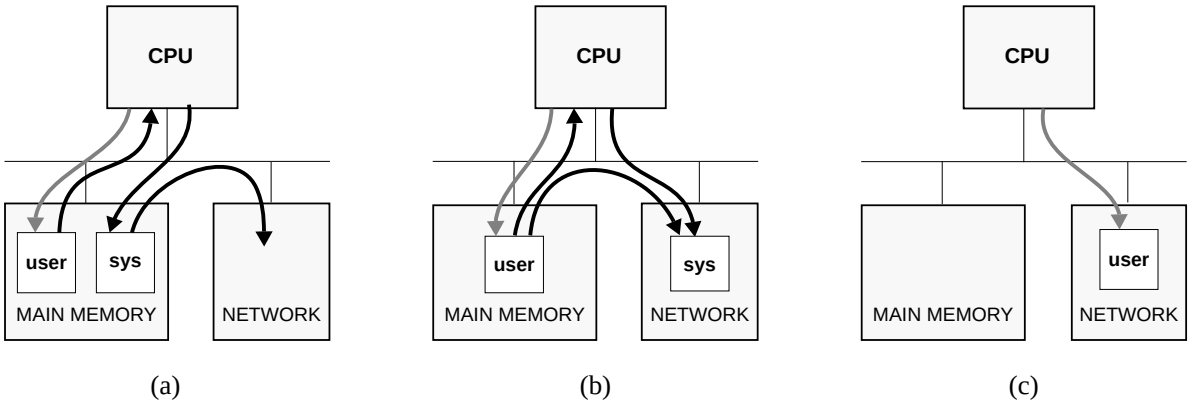

Figure 2: Design alternatives for the host-network interface

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.