Abstract

Nectar is a high-speed fiber-optic network developed at Carnegie Mellon as a "network backplane" to support distributed and heterogeneous computing. A distinguishing component of the Nectar network is a highly programmable network processor called the CAB. In contrast to most outboard protocol engines, the Nectar CAB has a flexible architecture, where almost all interactions between the network and the host are programmable. This structure allows arbitrary protocols to be implemented, evaluated, and utilized.

This paper describes the design, implementation, and usage of the Nectar CAB, and performance implications of its hardware features.

1 Introduction

The speed and memory size of small computer systems have increased dramatically over the past decade. Consequently, the demands of desktop workstations threaten to outstrip the capabilities of current networks that were designed more than a decade ago. For example, a pair of high-end workstations can potentially saturate an Ethernet.

The arrival of fiber optic technology has made it possible to implement high-bandwidth communications over long distances, leading to the development of a number of high-speed networks and the emergence of new network standards, such as FDDI and Fibre Channel [1, 2]. Moreover, large networks with high aggregate bandwidths can now be built using fast crossbar switches to overcome the bandwidth limitations of networks based on shared media, such as Ethernet and FDDI.

With the availability of powerful processor components, it has also become feasible to implement high-speed network processors to provide a faster interface to the network, and to off-load protocol processing from the host.

A significant potential application of high-speed networks connecting many high performance machines is the usage of groups of computers as a multicomputer. Finer-grained operations can be executed remotely, expanding the class of parallel algorithms which can be distributed over a network.

The objective of the Nectar network was to take advantage of recent technological advances to build an experimental high-speed network. The design of the Nectar network began in May 1988, and a 26-host system has been fully operational at the CMU campus since April 1990.

The Nectar architecture provides a general and systematic way to handle heterogeneity and task-level parallelism. A variety of existing systems can be connected to the network. The Nectar system software allows applications to communicate at a high level, without requiring each node to support a suite of network protocols; instead, protocol processing is off-loaded to a powerful network processor called the CAB (Communication Accelerator Board).

The Nectar CAB distinguishes the Nectar network from other high-speed networks because of its highly programmable architecture. In contrast to most network pro-
cessors which are primarily designed as outboard engines for a specific protocol, the Nectar CAB has a flexible architecture, where almost all interactions between the network hardware and the host are fully programmable. For example, the proposed VMP Network Adapter Board [8] is targeted towards a particular protocol (VMTP), and Chesson's Protocol Engine design [5] calls for a protocol (XTP) to be cast in silicon.

This paper discusses the Nectar CAB, including its design, usage, and performance. The rest of the paper is organized as follows. Section 2 gives an overview of the Nectar network, including the architecture of the switch and the current deployment. The design goals, architecture, and implementation of the CAB are described in Section 3. Section 4 explains how the CAB is used by systems and application software and gives a performance summary. Section 5 gives an analysis of the impact of various hardware features on performance and points out some improvements. Section 6 contains some concluding remarks.

2 Nectar Overview

There were three major technical goals for Nectar: heterogeneity, scalability, and low-latency, high-bandwidth communication. These aims follow directly from the desire to support a research environment where communication protocols for high-speed networks can be developed and evaluated. These high-level goals translated into the following basic architectural decisions. Support for heterogeneity was provided by using a powerful network processor that would attach to various hosts using a standard bus (VME-bus) and by implementing a common high-level software interface. The goal of scalability was addressed by allowing networks to be constructed using multiple switches interconnected in any arbitrary fashion. The goal of low-latency, high-bandwidth communications led to a crossbar switch based network topology where each host had a dedicated 100 Mbits/sec bandwidth to and from the switch, and a hardwired switch controller that can set up a connection in 700 ns.

2.1 System Architecture

A typical Nectar network is depicted in Figure 1. Two main components of the Nectar architecture are the HUB, which is a crossbar switch, and the CAB, which is the network processor used to attach a host computer to the network. A CAB is connected to a HUB using a pair of unidirectional fiber optic links each supporting 100 Mbits/sec bandwidth. Switch control and all routing and network-level error recovery functions are performed by software running on the CABs. The network can be extended arbitrarily by using multiple HUBs, where the HUBs can be interconnected in any desired topology using fiber optic pairs identical to those used for CAB-HUB connections. Passing through each HUB adds a minimum latency of 350 ns to the overall latency in end-to-end communications. To reduce bottlenecks in inter-HUB communication, two HUBs would normally be connected by multiple links.

2.2 HUB Overview

The HUB consists of 16 fiber ports, a 16x16 crossbar switch, and a hardwired HUB controller. It has a command set implemented in hardware that supports circuit switching, packet switching, multicasting, and multi-HUB connections. A HUB can set up a connection in 700 ns, and once a connection is established it can maintain the 100 Mbits/sec bandwidth of the fiber optic links.

Each fiber port contains fiber interface circuitry for optical to electrical conversion and vice versa, in addition to queues for data and commands. A fiber port can be connected to either a CAB or to another HUB fiber port by a pair of fiber lines.

The HUB has a 16x16x9-bit wide crossbar switch, which can connect the input queue of a port to the output queue of any other port. An input queue can be connected to multiple output queues for multicast. The crossbar switch has a cycle time of 70 ns, which is slightly faster (by 12 percent) than the speed of a fiber optic link, to ensure that the HUB can support the full bandwidth of the fiber optic links.

The HUB architecture supports an extensive command set (58 commands) to provide flexibility in implementing various datalink protocols. In addition to basic commands, such as open and close connections, the command set includes more elaborate commands such as "automatic retry with acknowledge" for connecting to busy ports, and commands for inquiries on HUB status.

In CAB-to-HUB or HUB-to-HUB communications, control bytes, which include commands and packet acknowledgements, are transmitted intermixed with the data stream to reduce communication latency. Control and data bytes are distinguished by a 9th bit available on the transmitter.
and receiver parallel/serial converter chips (Advanced Micro Devices TAXI's).

The HUB fiber port extracts commands from the incoming byte stream, and inserts replies to the commands in the outgoing byte stream. A hardwired central controller establishes and breaks connections through the HUB and controls the crossbar switch accordingly. A status table is used to keep track of existing connections and to ensure that no new connections are made to output queues that are already in use.

The HUB is implemented as two 14"x15" fiber interface boards plugged into a 14"x19" backplane. Each fiber interface board contains eight fiber ports. The backplane houses the crossbar switch and the HUB controller.

2.3 Status

The current Nectar system consists of 26 hosts connected to two HUBS. The hosts are distributed throughout the Carnegie Mellon campus. Additionally, one host resides at a site 26 km from the main campus. The hosts are primarily VME-based Sun workstations. The network also has connections to high-performance machines via Sun workstations, including the Warp [3] and iWarp [4] systolic array processors at CMU and a Cray-YMP at the Pittsburgh Supercomputing Center.

3 CAB Architecture

The main goal of the CAB architecture was to provide a flexible and highly programmable network processor. However, the need for flexibility had to be balanced by the desire for basic operations to be performed quickly. In general, the temptation to implement complicated protocol functions in hardware was avoided; only those functions that could not possibly be handled by software at fiber speed were cast in hardware. More emphasis was placed into producing a programmable board which would run efficiently at a high clock speed. In turn, a higher clock speed implies the CAB software can execute faster and more protocol functions can be satisfactorily implemented in software.

3.1 Major Blocks

There are three major blocks of the CAB: processing unit, network interface, and host interface. All three blocks have high-speed access to a 1 Mbyte packet memory as depicted in Figure 2. The processing unit consists of a SPARC processor, program memory, and support logic including counters, timers, and a serial port. The network interface consists of fiber optic data links, queues for buffering data streams, DMA channels for transmission and reception, and associated control and status logic. The host interface, designed for the VMEbus, includes slave ports for the host to access the CAB, DMA controller and bus master logic for the CAB to access VMEbus devices, and interrupt logic.

3.1.1 Processing Unit

The central processor is a Fujitsu SPARC running at 16.5 MHz. It has a program memory consisting of 128 Kbytes of PROM and 512 Kbytes of zero-wait-state SRAM. The program memory is separate from the packet memory because of bandwidth requirements. Since the SPARC CPU does not have an internal cache it requires high memory bandwidth from the program memory for instruction fetches. Therefore, implementing a single memory which would satisfy the bandwidth requirements of both the packet and the program memories was prohibitive. However, since the packet mem-
ory is also accessible by the CPU, it is possible for part of a program to reside in packet memory if the program memory is exhausted. The CPU can access virtually all resources of the CAB, including the network interface, the host interface, and various CAB registers. The registers include the status, control, interrupt, and interrupt mask registers for the interfaces and devices.

The processing unit also has a significant amount of hardware dedicated to programming support. These include three timers for monitoring and network timeouts, two 32-bit counters for finer precision timing, a serial port for debugging, and memory protection hardware to support multiple process domains. Although there is no virtual memory management on the CAB, memory protection is provided for every 1 Kbyte page of physical memory space, and up to 32 protection domains are available for each page. Four independent permission bits can be set for each page; three are used for protection against reads, writes, or execution, and one for generation of software traps. Memory protection can be enabled and disabled by the CPU. When an exception occurs, the faulting address is saved in an external register for the software debugger.

3.1.2 Network Interface

The network interface is further divided into two subsections: transmitter and receiver. There is almost no direct interaction between the transmitter and receiver; both units are independently controlled by the CPU.

The physical connection to the network is provided through Advanced Micro Devices TAXI parallel/serial converters, coupled with Sumitomo fiber optic data links. The TAXI chips use a 4-into-5 bit encoding scheme, so the effective network data rate is 100 Mbits/sec, given the raw transmission speed of 125 MHz.

The packet format used at the hardware level is depicted in Figure 3. The beginning is indicated by a Start of Packet (SOP) command byte, followed by an arbitrary number of data words and a 16-bit Cyclic Redundancy Code (CRC) tag before the End of Packet (EOP) command byte. Although the packet header format is not set in hardware, the Nectar datalink software uses a fixed header format consisting of three 32-bit words. As there are no constraints on the packet size, higher level software has the flexibility to use packet sizes best suited for application needs. In practice, packets range from a few words for interprocess communication in multicomputing applications to megabytes for transmitting images.

Flow control between CAB and HUB at the packet level is maintained by the use of SOP Acknowledge commands. When the CAB sends an SOP to the HUB, a flag is set in the CAB network interface. The mechanism allows only one unacknowledged packet at any time; the transmitter data queue is frozen until an SOP Acknowledge is received from the HUB. A similar mechanism is implemented on the HUB fiber ports. For every SOP received by the CAB, a corresponding SOP Acknowledge must be sent back to the HUB. Consistent with the CAB philosophy of high programmability, the transmission of SOP Acknowledgements is performed by the CPU under software control. This feature allows the datalink software flexibility in accepting new packets.

There is no flow control between the CAB and the HUB at the byte level. Implementing hardware flow control at a fine granularity would have reduced the peak transmission bandwidth. Instead there are input queues at the receive ports of the HUB and the CAB to provide buffering. On the CAB, the DMA channels between the packet memory and the network interface run slightly faster than the network speed (16.5 MBytes/sec compared to 12.5 MBytes/sec on the network). The difference in speed coupled with the queuing of data provides extra time for the CPU to initiate DMA transfers.

Transmitter

In addition to the fiber interface, the transmitter contains a 1Kx9-bit queue, hardware for CRC generation, and wired state machines to control transfers to and from the transmit queue. The transmitter is normally accessed by the CPU to send commands to the HUB and to write the header information of a packet. The packet data is transferred directly from the packet memory using the transmitter DMA channel.

Another function of the transmitter is the sending of SOP Acknowledgements to the HUB for received packets. It is critical to send the acknowledgements as quickly as possible to minimize communication latency and to implement efficient flow control. Therefore, acknowledgements are sent directly to the HUB, bypassing the transmit queue. SOP Acknowledgements can be piggybacked in the middle of an outgoing data packet or HUB command.

Receiver

Incoming data from the receiver fiber port is sorted into two queues. One queue is for data packets, the other for replies from the HUB to commands sent by the CAB. The receiver data queue is 2 Kbytes deep, while the reply queue is 1 Kbyte deep.
The depth of the receiver data queue determines the maximum time available for the CPU to process an incoming packet. The CPU must respond to the Start of Packet interrupt, interpret the header, and start a DMA transfer into packet memory before the queue overflows. Based on experience, it was found that a 1 Kbyte queue, which can be filled by the network in approximately 1350 CPU cycles, led to tight timing constraints for the datalink software. Increasing the queue size to 2 Kbytes provided a more sufficient period of time to respond.

Similar to receiving data packets, the CPU is interrupted upon receipt of replies from the Hub. The replies are in response to commands sent from the CAB to the Hub, such as for establishing connections or inquiries on Hub status. The CPU dequeues replies from the queue and acts on them appropriately. Although the queueing requirements are less stringent for the replies than for data, a 1 Kbyte queue is provided.

### 3.1.3 Host Interface

The CAB host interface is designed for the VMEbus. The VMEbus was chosen because it is widely used in existing workstations, especially in the Sun workstations at CMU. Although recent workstations use faster buses which would be a better match to the fiber bandwidth, the VMEbus was considered the most feasible alternative at the time of design.

The CAB can be either a bus slave or master. As a slave, any location in the CAB address space can be read or written via the VMEbus. The CAB slave interface supports byte, half-word and long-word reads and writes. If there is a conflict with the CAB CPU in accessing a resource, the CPU is halted while the host interface logic accesses the corresponding memory or device. Since the majority of accesses from the host interface are to the packet memory, a separate port was provided to the packet memory in order to avoid halting the CPU. In addition to preserving processor bandwidth, the dedicated port also allowed us to implement faster host accesses to the packet memory than would have been possible otherwise.

As a master, the CAB can access the VMEbus directly by the CPU or by on-board DMA. The CPU accesses are used primarily to set up transfers between the CAB and an external device such as a frame buffer. The DMA channel is used for the actual transfer of data between the device and the packet memory. During DMA, the bus transfers can be 32 or 16 bits wide, and the CAB can transfer blocks of data to or from a constant VMEbus address. The constant address feature was added particularly to support access to devices that have a FIFO-type interface. During DMA transfers, the CAB uses a Release-On-Request bus policy to reduce the arbitration overhead.

The host can interrupt the CAB and vice-versa. An interrupt is generated on the CAB by sending a VMEbus interrupt at a hardwired level or by writing to a reserved address in the CAB address space. The latter feature was added because the Sun VMEbus interface lacks a mechanism for generating bus interrupts. The CAB can send a VMEbus interrupt to the host on a hardwired level by writing an interrupt vector to a reserved address.

### 3.2 Implementation

The CAB is implemented on a 9U VMEbus board (approximately 14"x15"). There are 360 active components on the board; all are commercially available off-the-shelf components. The major components include a 20 MHz SPARC processor, two 16-bit ALUs for the DMA engine, and 53 high-speed (35 ns) SRAM chips. Most of the logic is implemented using PALs; there are a total of 102 PALs of various kinds on the board. The remaining components include 168 registers and buffers, 3 queues, 2 125 MHz TAXI chips, 2 optical data links, 3 oscillators, and about 40 miscellaneous chips (timers, UART, etc.). Figure 4 shows a picture of the CAB.

![Figure 4: Picture of CAB](image)

### 4 Usage and Performance

In this section, we give an overview of how the CAB is used by systems and application software, the performance of basic network protocols, and a summary of distributed computing applications using Nectar. The details can be found in several other papers on Nectar [7, 9].

#### 4.1 Systems Software

To implement a usable system, layers of software were constructed on top of the CAB hardware, both resident and on the host. Almost all of the system software was written in the C programming language. User application code is also written in C and accesses system services through linked libraries. The CAB kernel provides a runtime system for multiprogramming using a threads package derived from...
Mach C threads [6]. Threads support multitasking so the CAB can execute multiple activities concurrently, including network interrupts, transport protocol processing, and application-specific computation. The CAB kernel also includes a debugger for software development.

The datalink layer sits above the CAB network hardware and provides basic services to send packets point-to-point. It supports both circuit and packet switching, normally using packet switching for small packets and circuit switching for large packets.

Several transport protocols have been implemented on the CAB including TCP/IP and a set of Nectar-specific protocols. The Nectar specific protocols are datagram, reliable message, and request-response protocols. The datagram protocol has low overhead but does not guarantee packet delivery. The reliable message protocol uses acknowledgements and retransmissions to guarantee packet delivery, and the request-response protocol supports client-server interactions such as remote procedure calls.

The application interface is provided by Nectarine, a programming library developed for Nectar. Nectarine provides a simple communication abstraction to the programmer where applications consist of tasks communicating by messages. Tasks can reside on any host or CAB on the network, and Nectarine presents the same interface regardless of the location. It is implemented as a library linked into an application’s address space, and uses a mailbox mechanism for sending and receiving messages between tasks. The buffer space for mailboxes is allocated in CAB packet memory, where host threads can build and consume messages.

4.2 Usage

The flexibility of the CAB hardware and software architecture allows it to be used in various levels by changing the interface the CAB presents to the host. Three such interfaces are summarized below, ordered in increasing degree of CAB functionality.

The Nectar network can be used as a conventional, high-speed LAN by treating the CAB as a network device. The advantage of this approach is binary compatibility: all familiar network services are immediately available using a CAB device driver on the host.

The CAB can be used as a protocol engine by off-loading transport protocol processing to the CAB. Various transport protocols implemented on the CAB are described in the previous section. Several interfaces are being developed for the host, including the Nectarine interface, the Berkeley socket interface, and the Mach interprocess communication interface.

At the application level, the CAB can be used as a communication engine since part of the application code can be executed on the CAB. The CAB and its runtime system are more flexible than many protocol engines. Distributed applications on Nectar often use both the host and the CAB, effectively using the CAB as an application-specific network interface.

4.3 Performance

For moderately large packet sizes (greater than 8 Kbytes), communication between two CABs is near the raw network speed. The throughput for CAB-to-CAB communication using the Reliable Message Protocol (RMP) and TCP (without checksum) are both slightly greater than 10 Mbytes/sec. For smaller packet sizes (1 Kbyte), the throughput is 4.5 Mbytes/sec for RMP and 3.5 Mbytes/s for TCP.

Due to limitations in the Host-CAB VMEbus interface (to be explained in Section 5), the performance numbers for host-to-host communication are significantly lower than for CAB-to-CAB. For packet sizes of 8 Kbytes, the maximum bandwidth between two hosts using the RMP is 3.5 Mbytes/sec, while the maximum TCP bandwidth is approximately 3 Mbytes/sec. For packet sizes of 1 Kbyte, the transmission rates are 3 Mbytes/sec for the RMP and 2 Mbytes/sec for TCP.

The round trip latencies for host-to-host communication are expectedly greater than those for CAB-to-CAB. The host-to-host round trip times for RMP and UDP are 414 and 842 microseconds respectively, while for CAB-to-CAB they are 241 and 536 microseconds.

To gain insight into the sources of the inter-host latency, we can look at the case of one-way host-to-host datagram, which is the simplest of the Nectar transport protocols. The more complicated protocols, such as RMP and UDP, have higher latencies because of the additional services they provide, such as reliability and enhanced addressing. The minimum latency of a one-way host-to-host datagram is 163 microseconds (round trip time is 325 microseconds). The breakdown is as follows: host builds message on CAB (20μs), host informs CAB to send message (60μs), transport layer (datagram) protocol on CAB (5μs), datalink sends packet (10μs), SOP interrupt on receiving CAB (12μs), datalink receives packet (18μs), passes message to host (10μs), and receives message on host (20μs).

4.4 Applications

Examples of applications already implemented on Nectar are: Noodles, a solid modeling system; COSMOS, a re-implementation of a switch-level logic simulator; and Pollution Modeling, which simulates the movement of smog over a large region. These applications use common parallel processing paradigms, such as divide-and-conquer and task-queue models, and execute on up to 20 Sun workstations. Further details of these applications and their performance can be found in [9].

Several other applications that can make good use of the
flexibility of the CAB are under development, including the Camelot distributed transaction system, the Mistral ray-tracing system, and the Paradigm database for vision.

5 Analysis

Different features of the CAB hardware impact the system performance at different layers of the Nectar software stack. At the datalink layer, the particulars of the network interface are most dominant. The transport protocol layers are influenced most by the CAB CPU execution speed and context switch time. The Nectarine level host software and application layers are impacted significantly by the speed of the VMEbus interface. Next, we discuss how some specific features of the CAB architecture impact the performance in these three layers. We also mention several possible improvements for future architectures.

5.1 Datalink Layer

The hardware feature which contributes most to the performance of the datalink software is the DMA channels between the network interface and the packet memory. Without the DMA channels, it would not have been feasible to keep up with the speed of the fiber with a single CPU, as the CPU would have to transfer every byte between the network interface and the packet memory, in addition to other processing tasks. DMA transfers are flow-controlled in hardware and run 30% faster than the network speed. The CPU can be interrupted at the completion of DMA, or it can poll on a status bit.

Certain modest hardware features can save several cycles from tight loops. For example, flow control in accessing the queues saves cycles by relieving the CPU from polling the queue status. For example, when a word is written to the transmit queue, the CPU is stalled if the transmitter logic is busy queuing the previous word into the queue. Without this feature, it would be necessary to poll a status bit until the transmitter is ready for another word.

Another way to save CPU cycles is to make hardware functions more atomic. For example, in the first version of the CAB it was necessary for the CPU to explicitly start a DMA transfer. In the second version, this operation was eliminated by having the DMA start immediately after writing the DMA counter and start address.

However, there is often a temptation to carry this process further to implement higher-level datalink functions in hardware. For example, it was considered to automatically start the DMA of incoming packets, extracting the start address and length from the datalink header fields by the network interface. This hardware implementation would have eliminated the latency for responding to the SOP interrupt by the CPU, which is several microseconds. On the other hand, it would have removed the flexibility of the CAB by casting a particular datalink packet format in hardware. In addition, memory management and handling exceptional cases would have been more difficult. For example, a header checksum would have to be included and verified in hardware to prevent incorrect transfers to the packet memory. Therefore, these functions were left to the software.

5.2 Transport Layer

Transport layer protocols are implemented on the CAB using C threads, which allow concurrent programming using multiple lightweight processes. The use of threads requires the CAB to change context when a thread yields or is preempted. The context switch time on a CAB is rather high, about 20 microseconds on average. This is primarily due to the large number of register windows on the SPARC that need to be saved on a context switch. The amount of processor state to save is 132 words of scalar registers plus 10 registers of miscellaneous state. In retrospect, the SPARC was not the best choice for a system where fast context switching is an important feature. However, the decision to use the processor was made due to its availability, both in terms of silicon and support tools.

Since the CAB was designed as a flexible platform, there are no hardware features to support a specific transport protocol such as TCP. When implementing TCP on the CAB, it was observed that calculating the TCP checksum with the CPU is the biggest bottleneck. For TCP with the 16-bit checksum calculated by the CAB CPU, the maximum transmission rate between CABS is 4.5 Mbytes/sec. Without calculating the TCP checksum the rate is about 10 Mbytes/sec. Thus if the checksum was calculated in hardware, the TCP performance would more than double. In general, it would be beneficial to perform all mechanical functions that require inspection of each word of data, such as packet checksum, in hardware. Otherwise, providing hardware support such as DMA to speed up the datalink protocols becomes relatively insignificant.

5.3 Host Application Layer

Applications and the underlying software resident on the host need to access the CAB over the VMEbus. The bandwidth of data transfers over the VMEbus turned out to be the most limiting factor in the performance of host-to-network communications.

Using a raw access test, a Sun4/330 can write the CAB packet memory at a peak bandwidth of 4.4 Mbytes/sec, which is approximately 900 ns to transfer a 32-bit word. However, the low bandwidth is partially due to the limitation of the Sun accessing a VMEbus device. For comparison, the peak write bandwidth to zero-delay VMEbus memory using the same test would be 6.4 Mbytes/sec. In contrast, the main memory of the Sun, which is not on the VMEbus, can be accessed at a peak bandwidth of 11.8 Mbytes/sec. The access time to the CAB packet memory from VME is 290 ns, which is
comparable to the speed of a regular external VME memory.

The CAB was designed with a VMEbus interface primarily for practical reasons including widespread availability, board size, and power budget. However, it has become clear that the VMEbus interface is not adequate to keep up with the 12.5 Mbytes/sec bandwidth of the fiber interface. Using a synchronous I/O bus closer to the MMU (such as the Sun SB) could support significantly higher bandwidths and would be more appropriate for a high-speed network processor such as the CAB.

One significant factor contributing to host-to-CAB communication performance was mapping the CAB memory to the host address space. This allowed the application processes to construct and consume messages directly in the CAB memories, eliminating an extra copy between the host and the CAB memories.

In general, the Nectar network was designed with the intention of supporting distributed computing applications. Since the CAB is a powerful processor with sufficient programming support, it is possible to run application programs on the CAB itself. For some applications it is possible to run most or all of the application program on the CAB. However, most applications need the resources of a general-purpose host including a large physical memory and a file system. In several distributed computing applications on Nectar, parts of the code which are more intensive in communication were migrated to the CAB to reduce the host-CAB communication bottleneck [9].

6 Concluding Remarks

Nectar has proven to be a valuable tool for research in high-speed networks and distributed computing. Critical to the success of Nectar is the CAB's flexible architecture. It allowed development and evaluation of various alternatives in using a high-speed network, ranging from low-level datalink protocols to high-level interfaces for distributed computing.

Based on the CAB experience, some insight was gained into the design of programmable network processors, including the finer interactions between hardware and software, and the performance implications of certain hardware features. This insight will be useful in designing future network processors, especially for the development of the gigabit Nectar which is currently under way in a joint project between CMU and Network Systems Corporation.

Acknowledgements

The Nectar project involves a group of people at CMU led by H.T. Kung. We especially would like to acknowledge the work of Emmanuel Arnould toward the design and implementation of an earlier version of the CAB. The following people have significantly contributed to the development of Nectar hardware and software: Matthieu Arnould, Michael Browne, Bernd Bruegge, François Bitz, Eric Cooper, Fred Christianson, Robert Sansom, Peter Steenkiste, and Brian Zill.

References