HP-LAM: an Implementation of Generic Active Messages for the Hewlett-Packard 9000/J200 Workstation

by

Cedric Krumbein

B.S.E.E. (California State University, Fresno) 1981
B.S.C.S. (University of Washington) 1993

A report submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Prof. David E. Culler, Chair
Prof. David A. Patterson

December 19, 1996

Abstract

HP-LAM: an Implementation of Generic Active Messages for the Hewlett-Packard 9000/J200 Workstation

by

Cedric Krumbein
Master of Science in Computer Science

University of California at Berkeley

Prof. David E. Culler, Chair

This report describes the **Myrinet interface card**, a card that interfaces the Hewlett-Packard 9000/J200 workstation to the Myrinet local-area network, and **HP-LAM**, an implementation of Generic Active Messages (GAM) on that platform. It describes the performance of HP-LAM in terms of LogP parameters and in terms of execution time for the Split-C application `radix sort`. It shows how HP-LAM's performance is limited by inefficiencies in transferring data over the peripheral bus. It compares the performance of HP-LAM to that of Sun-LAM, a similar implementation of GAM on the Sun Microsystems SparcStation-20 and Ultra-1 workstations. The comparison shows that the performance of HP-LAM is comparable to that of Sun-LAM on the SparcStation-20 but is poorer than that of Sun-LAM on the Ultra-1.

Contents

List of Figures

List of Tables

Acknowledgements

It's natural in a project that runs for two years for the list of people who contributed to it to grow alarmingly long. At U. C. Berkeley, first let me thank my committee. Thanks to my advisor, Prof. David E. Culler, and to Prof. David A. Patterson for their timely advice and support. In my office, special thanks to Rich Martin for his advice and insights into the inner workings of Active Messages and his experience with Myrinet. Thanks also to Ken Lutz and to two undergraduates, Sandeep Chatterjee and Young Cho, for their assistance in the design of the hardware.

At Hewlett-Packard Laboratories, many thanks to the Hamlyn team, Greg Buzzard, David Jacobson, and Scott Marovich for their advice and support. Special thanks to Scott, whose extensive knowledge of HP-UX was indispensable.

At Myricom, thanks to Bob Felderman and Wen-King Su for their answers to dozens of questions on the inner workings of Myrinet.

Finally, there's no question who provided me the most support in this project, even though she isn't even a computer techie. Endless thanks and love to my wife, Nancy Hosken, who kept on believing in me whenever I had my doubts.

Chapter 1

Introduction

Until recently parallel processing has been supported primarily by massively parallel processors (MPPs). An MPP is essentially a set of processing nodes connected by a high-speed network, each node executing one thread of the parallel program. MPPs have traditionally fallen into two groups. (1) In a **shared memory** machine, every node "sees" the same memory image. The shared memory may be implemented in hardware as a single monolithic block, or it may be distributed among the nodes, each node owning a piece of the total memory. The nodes communicate either by accessing common memory locations or by passing messages. (2) In a **message-passing** machine, each node maintains its own local memory image, and the nodes communicate solely through message-passing.

The performance of the network in an MPP is crucial, whether it is used to support the shared-memory image or to pass messages. MPP producers have thus pioneered a number of innovations in high-speed networking, such as the use of fast parallel switches, topologies such as fat trees and hypercubes, and the use of fast, low-overhead network protocols. The recent emergence of high-speed commercial networks such as ATM and Myrinet [24], and of fast network protocols such as Active Messages [30], Fast Messages [21], Hamlyn [6], and U-Net [5], is due in part to innovations that first appeared in MPPs.

Ironically, these same high-speed networks, and the continuing phenomenal growth in the performance of commodity processors, are helping to bring about the demise of MPPs. Networks of workstations, or NOWs, are beginning to replace MPPs as the platforms used to support parallel processing. A NOW is a collection of commodity workstations connected by a high-speed network using a fast network protocol. Special-purpose software provides the support for managing parallel programs, such as scheduling the nodes, providing a common file system, *etc.*

In the Network of Workstations project at U. C. Berkeley [2], we have developed three generations of NOWs. Each version has used a different combination of workstations, operating systems, and network hardware and protocols. The prototype version, **NOW-0**, was built using Hewlett-Packard (HP) 9000/735 workstations. Fiber-Distributed Data Interface (FDDI) was used for the high-speed interconnect, and Generic Active Messages (GAM) [9] was used for the fast network protocol [18]. **NOW-1** used Sun SparcStation-10 and -20 workstations, the Myrinet local-area network, and an implementation of GAM for that platform called Sun-LAM. **NOW-2** is currently being built using Sun Ultra workstations, Myrinet, and Active Messages-2 (AM-2) [16].

We faced a number of choices in selecting the components for each generation of NOW. For NOW-1, Myrinet and GAM were selected early for the network hardware and protocol. The choice for the processing node was narrowed down to the Sun SparcStation [28] and the HP 9000/J200 [12]. However, at that time no HP workstation supported an interface to Myrinet. This report describes my effort to develop Myrinet interface hardware and GAM software for the HP 9000/J200.

In this report I describe the **Myrinet interface card**, a card I developed that interfaces the Hewlett-Packard 9000/J200 workstation to Myrinet; and **HP-LAM**, an implementation of Generic Active Messages on that platform. I describe the performance of HP-LAM using microbenchmarks, LogP parameters, and in absolute execution time for the Split-C application `radix sort`. I show how the performance of HP-LAM is limited by inefficiencies in transferring data over the 9000/J200's peripheral bus.

Our initial motivation for developing the card and HP-LAM was to make the 9000/J200 a potential platform for NOW. This has become less important since Sun workstations were selected as the platforms for NOW-1 and NOW-2. However, another set of issues that we have yet to tackle in the NOW project involves building a NOW using *heterogeneous* workstations. To date each generation of NOW has been built using the same model of workstation for every processing node. HP-LAM provides the opportunity to investigate the issues surrounding the building of a heterogeneous NOW, in this case a network made up of Sun and HP workstations.

This report is organized as follows. The remainder of this chapter provides background on the Active Messages protocol and the Myrinet local-area network. Chapter 2 describes the **hardware** design of the Myrinet interface card for the HP 9000/J200, and chapter 3 describes the design of the HP-LAM **software** implemented on this platform. Chapter 4 describes the **performance** measurements done on HP-LAM and compares them to GAM implementations on other platforms, especially Sun-LAM. Chapters 2–4 each end with a discussion of the issues faced in each effort and the lessons learned. Finally, chapter 5 provides an overall **summary** of the project.

1.1 Active Messages

Active Messages, or AM [30], is a fast network protocol and API developed to support fast messing-passing in an MPP or a NOW. AM is able to deliver messages between nodes with less overhead and latency than can be provided by more traditional network protocols such as TCP/IP. AM is now also being used to provide more traditional network services such as Berkeley sockets within a local-area network (LAN), again with greater performance than provided by TCP/IP [23].

AM is most commonly used as the communication layer of single-program, multiple-data (SPMD) parallel program. In this setting each node of the parallel program runs on a separate processor, and the program image is identical on every node. An active message can be viewed as a restricted, lightweight remote procedure call. Each active message consists of two parts, a *request* and a *reply*, each specifying a *handler function* and arguments. As an example, assume one node (the *source*) sends a message to a second node (the *destination*).

This first leg of the message is the *request*. Upon arriving, the destination node executes the specified *request handler* using the supplied arguments. The request handler returns a *reply* to the source node; when it arrives, the source node executes the specified *reply handler*. An active message is considered "delivered" when the reply handler completes executing.

AM achieves high performance and low overhead in a number of ways. A program using AM sends and receives messages by reading and writing them directly into the network interface at user-level. An active message does not have to travel through a protocol stack the way TCP/IP data does. With the operating system and the stack removed from the message path, a message crosses fewer boundaries and is copied fewer times. (The operating system *may* participate, however, in other networking operations such as initializing the network interface, setting up connections, *etc.*)

Several versions of AM have been developed. The specific version of AM implemented in HP-LAM is Generic Active Messages, or GAM [9]. GAM allows a single user per node access to the network. It is implemented as a user-level library that is linked into the parallel program. The GAM specification lists a set of functions that all implementations must provide, which enables a parallel program to be ported to any platform that supports GAM. GAM guarantees reliable message delivery, but it does *not* guarantee in-order delivery.

GAM is *not* a general-purpose network protocol like TCP/IP. It is intended to be used within an MPP or a NOW by a single program.

Beyond these features common to all implementation of GAM, each implementation may use any policy that best suits the particular hardware and software constraints of the system. For example:

- **Routing.** Each node in an n-node parallel program using GAM is identified by an integer $0..n$-1. The nodes send messages using these identifiers. Every GAM implementation translates these integers into routes; how this is done depends on the routing policies of the network.

- **Reliability.** To provide reliable delivery, GAM implementations fall into three general classes: (1) MPP networks such as those of the CM-5 and the Meiko CS-2 are considered *extremely reliable*. In these networks errors are so infrequent that when one actually does occur, the parallel program is considered corrupted and simply aborts. The GAM software need not include any error-correcting routines. (2) Traditional workstation networks such as Ethernet are considered *unreliable*. GAM software implemented on such a network must include error-detecting and -correcting routines; further, errors are assumed to occur frequently enough that the routines must be highly optimized. (3) Between these two are *nearly reliable* networks such as FDDI and Myrinet. In these networks errors can occur and thus the GAM software must include error-detecting and -correcting routines, but errors are considered to be infrequent enough that the routines need not be highly optimized. (HP-LAM assumes a nearly reliable network.)

To date GAM has been implemented on the following MPP platforms: the Thinking Machines CM-5, the nCUBE/2, the Intel Paragon, the Meiko CS-2, and the Cray T3D.

GAM has also been implemented on the following commodity workstations and local-area networks: the Hewlett-Packard 9000/735 workstation using FDDI, and the Sun Microsystems SparcStation and Ultra-1 stations using Myrinet.

1.2 Myrinet

The Myrinet local-area network was developed by Myricom, Inc. [24] Myrinet is one of an emerging group of high-speed packet-switched data networks. Unlike other such networks (*e.g.*, ATM), Myrinet switches are connectionless and stateless. All of the state in the network is in the interfaces.

The Myrinet interface is a general-purpose processor that allows almost any protocol and API to be supported. The interface includes local memory for buffering and a DMA engine to transfer bulk data between the interface and host memory.

A Myrinet switch is a stateless crossbar. Switches support cut-through message-forwarding and use relative, source-based routing. Switches are available in several sizes: 4, 8, or 16 link ports.

Myrinet links are full-duplex, parallel electrical cables capable of carrying 80 MB/sec of bandwidth in each direction.

Myrinet uses token-based packet framing and flow control. The maximum size of a Myrinet packet is unspecified, though the maximum length used by HP-LAM is 4 kilobytes.

Chapter 2

Hardware Design

This chapter describe the the design of the Myrinet interface card; that is, the hardware developed to support HP-LAM on the Hewlett-Packard 9000/J200 workstation.

A block diagram of the 9000/J200 is shown in Figure 2.1. For the purposes of this report, each 9000/J200 consists of one or two HP PA-7200 PA-RISC host processors [13] and 32–1024 megabytes of main memory connected across a Runway system bus. The Runway bus is connected to two GSC+ busses [11] by a U2 Bus Adapter chip [29]. Only one of the two GSC+ busses in the 9000/J200 has card connectors that allow the installation of optional peripheral cards such as network interfaces. The other GSC+ bus is used to connect to other peripheral devices such as the console TTY line and the Ethernet adapter.

Two similar Myrinet interface cards were developed for use in the 9000/J200's GSC+ bus. I developed a card as part of my graduate research at U. C. Berkeley; hereafter I refer to it as the **Berkeley card** [15]. The Berkeley card was designed using the Myricom, Inc., SBus-Myrinet card [24] as a model; most components in the Berkeley card have direct counterparts in the SBus card. After presenting my proposed design of the Berkeley card to Myricom and HP Labs, they refined the design and jointly developed the GSC/Myrinet Device Adapter card [17], hereafter referred to as the **Myricom card**. Unfortunately, the Berkeley card did not achieve functionality; therefore the Myricom card has been used in this report for software development and performance measurements. For the remainder of this report, I refer to either card generically as the **Myrinet interface card**, and I indicate features specific to either card when relevant.

The Myrinet interface card can only be used in the HP 9000/J200. The card was designed to install in the 9000/J200 's GSC+ bus. HP uses the GSC+ bus in other systems; however, the card form factor is unique to each system in which the bus is used. GSC+ cards in the 9000/J200 conform to the modified EISA card form factor. The same card design could be adapted to any other HP system that uses the GSC+ bus by re-laying the card using that system's form factor.

In the HP PA-RISC I/O architecture [22], each peripheral device is mapped to a fixed location in the host processor's physical address space depending on the bus and slot in which it is installed. This allows the host processor to communicate with any peripheral device by performing memory-mapped reads and writes to it. In PA-RISC nomenclature these are called **Direct Input/Output** or **DIO** transactions. Peripheral cards may also

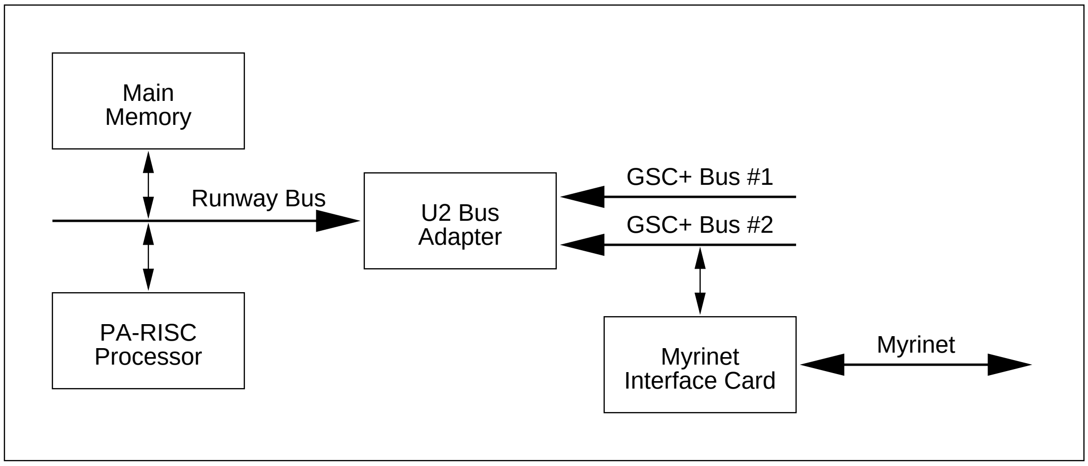

Figure 2.1: **HP 9000/J200 Block Diagram including a Myrinet Interface Card.** The 9000/J200 consists of one or two HP PA-7200 PA-RISC host processors and 32-1024 megabytes of main memory connected by a Runway system bus, two GSC+ busses (one with card connectors for peripherals), and a U2 Bus Adapter connecting the busses. The Myrinet interface card installs in a GSC+ card connector.

read and write to the main memory directly; these are called **Direct Memory Access** or **DMA** transactions.

Each peripheral device must also support a set of IODC registers. These are registers that allow the host processor to detect the presence of a device during system initialization and perform any device-specific configuration. During initialization the host processor scans the entire I/O address space for peripheral devices. A device announces its presence by responding to a read request at its assigned address. The host then reads the peripheral's IODC registers to identify it, read any configuration parameters, *etc.*

Graphics System Connect (GSC+), also known as the HP-HSC bus [11], is a 40-MHz peripheral bus. Address and data are multiplexed onto the same 32 lines. GSC+ devices may perform transactions with each other, or they can send and receive transactions from the host processor across the U2 Bus Adapter.

GSC+ supports three types of transactions: writes, connected reads, and pended reads. (1) In a **write** the address is placed on the bus followed by one or more words of data. The device that owns the address must return an acknowledgement. (2) In a **connected read** the address is placed on the bus, and the device that owns the address must return data within a fixed number of cycles; the bus is idle until the data is returned. (3) A **pended read** is similar to a connected read, except that other transactions may take place before data is returned. For all three types of transactions, up to eight words of data may be transferred in a single transaction. (The Myrinet card supports writes and connected reads, but it does not support pended reads.)

The U2 Bus Adapter chip maps subranges of the two GSC+ busses' address ranges into the Runway bus' address space, allowing the host processor to perform DIO reads and

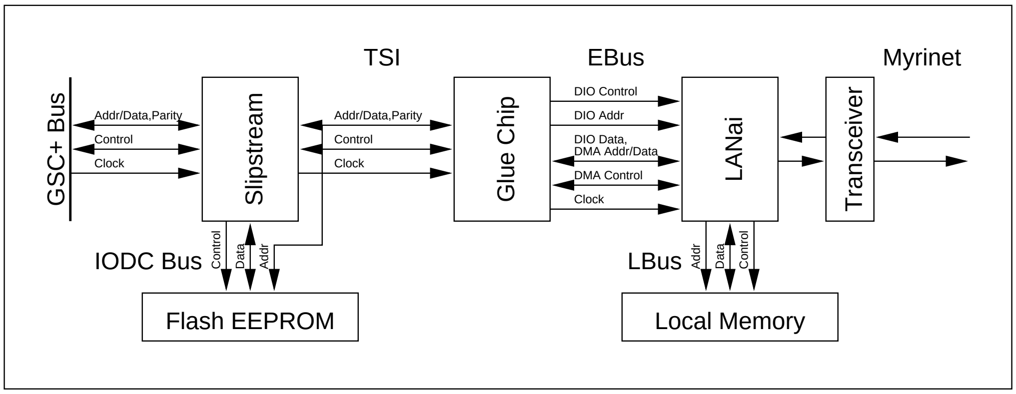

Figure 2.2: **Myrinet Interface Card Block Diagram.** The LANai is the central processor, network transmitter and receiver, and local memory interface for the card. The Slipstream and glue chips form a two-stage translator between the system's GSC+ bus and the LANai's EBus. The local memory holds network messages, the LANai control program (LCP), and routing tables.

writes to GSC+ addresses that fall within those subranges. Similarly, the U2 also maps subranges of the Runway bus' address range into each GSC+ bus' address range, allowing GSC+ cards to perform DMA accesses to selected main memory addresses. These mappings are written into the U2's page tables in main memory by the host processor.

The U2 also performs the GSC+ bus arbitration. A device asserts a Bus Request signal to obtain control of the bus, and the U2 replies with a Bus Grant when the bus is available. A device may perform several transactions within a single Bus Request/Bus Grant cycle; each device is responsible not to abuse this feature to monopolize the bus. The Myrinet card uses this feature to perform bulk DMA transfers to and from main memory. A single DMA transfer may move thousands of words of data, but it consists of a series of eight-word bursts.

The Myrinet interface card is designed to receive commands and data from the host processor in the form of DIO transactions. The card can also initiate DMA transactions to transfers data to and from main memory by using an on-card DMA engine. A block diagram of the card is shown in Figure 2.2.

The LANai processor chip [20] is the card's central processor and network protocol engine. The card has local memory for holding the LANai Control Program (LCP), network messages, and data. The card uses a Slipstream chip [26] and a field-programmable logic device (FPGA), or "glue" chip, to form a two-stage translator between the GSC+ bus and the LANai's EBus. Finally, the card has a set of transceiver chips for transmitting and receiving from the Myrinet link.

The rest of this chapter describes each of these components in more detail.

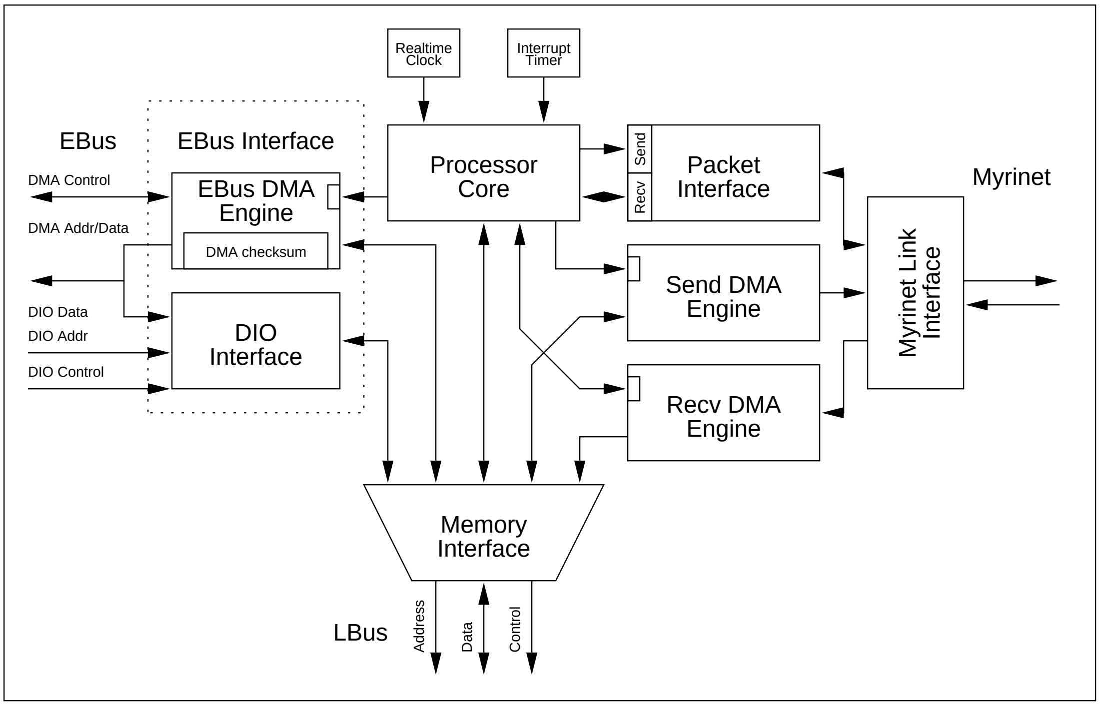

Figure 2.3: **LANai Block Diagram.** The LANai is a general-purpose processor with additional features to support network processing.

2.1 LANai

The heart of the Myrinet interface card is Myricom's LANai processor chip. The LANai is a general-purpose processor with additional functional units to support network processing. Its purpose is to execute a program (called the LANai Control Program, or LCP) to perform the functions required to support the desired network protocol; *e.g.*, sending and receiving messages, checking message integrity, managing routing tables, and so on. This general-purpose design of the LANai gives it the flexibility to support a wide range of protocols and APIs. A block diagram of the LANai is shown in Figure 2.3.

The LANai interfaces to three busses. (1) The **External Bus**, or **EBus**, is the LANai's interface for communicating with the rest of the system. The LANai receives DIO transactions from the host processor via the EBus, and it initiates DMA transactions across the EBus to access host memory. (2) The **Local Bus**, or **LBus**, is the interface to the card's local memory. The LANai manages all local memory accesses. (3) Finally, the LANai interfaces to the **network link**. The LANai sends and receives network messages using its packet interface and its send and receive DMA engines.

The LANai contains the following functional units:

- The **processor core** is a general-purpose processor that was derived from the Caltech Mosaic processor [25]. The core executes the LCP, which allows the designer to implement virtually any desired network protocol and API.

- The **EBus DIO interface** passes host-initiated DIO transactions from the EBus to the local memory. This makes the local memory look like part of the host processor's physical memory space, which enables the host processor to use DIO transactions to download the LCP, download outgoing messages, and read received messages directly from local memory.

- The **EBus DMA engine** allows the LANai to initiate DMA transactions to transfer data directly between main memory and local memory. The DMA engine automatically generates a checksum for transfers from local to main memory, allowing received messages to be validated.

- The **Myrinet Link interface** is a full-duplex interface; *i.e.*, it allows messages to be sent and received at the same time. The link interface contains all of the control needed to support the Myrinet signalling protocol; the only other hardware needed on-card are the transceiver chips (sec. 2.5). For outgoing messages, the interface formats the data into the proper network signalling format and inserts control symbols to frame the packets and provide flow control. For incoming messages, the interface extracts the data from the incoming packets. The interface appears to the processor core as a set of registers that the core can write to to send messages and read to receive messages.

- **Send and Receive DMA engines.** These allow the processor core to transfer large messages between the network links and local memory using only a few setup commands. The engines can operate simultaneously, their performance limited only by the links and by contention for the local memory.

- **LBus interface:** manages all accesses to the local memory. Accesses may originate from the host processor in the form of DIO transaction across the EBus, from the EBus DMA engine as it performs LANai-initiated DMA transfers to or from main memory, from the Send and Receive DMA engines as they perform DMA transfers to and from the network link, and from the processor core as it fetches instructions, and loads and stores data. The LBus interface allows two local memory accesses per clock cycle, and it prioritizes these accesses as follows:

Highest:	EBus DIO and DMA
	Receive DMA
	Send DMA
Lowest:	Processor core

The Berkeley card uses LANai ver. 2.3, and the Myricom card uses LANai ver. 4.0. The main differences between these two versions are that (1) their processor cores support difference instruction sets; (2) ver. 2.3's processor core is 16-bit, while ver. 4.0's is 32-bit; (3) ver. 2.3 can address 128 kilobytes of local memory, while ver. 4.0 can address 512 kilobytes; (4) ver. 4.0's EBus includes parity with address and data, but ver. 2.3's does

	Berkeley card	Myricom card
LANai version	2.3	4.0
LANai word size	16 bits	32 bits
Local memory size	128 kB	512 kB
Network signalling	PECL	LVDS
Network driver chip(s)	5 AT&T 41MM transceivers	Myricom-custom LVDS transceiver
EBUS parity generator	In glue chip	In LANai
Interrupt mask registers	None	3 in glue chip

Table 2.1: **Differences between the Berkeley and Myricom Cards.**

not; (5) the two versions support different network signalling protocols (sec. 2.5); and (6) the two versions have different pin layouts. Table 2.1 summarizes the differences. Despite these differences, the same LCP can run on both versions after re-compilation, and both can support HP-LAM.

The EBus is the LANai's interface for communicating with the rest of the system. Since the first generation of Myrinet interface cards were designed to interface to the Sun SBus, Myricom designed the EBus to be very similar to SBus. EBus and SBus can be connected together with very little additional hardware. Fortunately, Myricom also designed EBus to be flexible enough that it can be made (with enough effort) to interface to other busses that are similar to SBus. The majority of the effort in developing the Myrinet interface card was in designing this interface.

The following are the main challenges in interfacing EBus to GSC+:

- **GSC+ Electrical Requirements.** All GSC+ signals operate at 3.3 volts, except for the clock signals which use differential ECL. The LANai and the EBus, however, operate at 5 volts.

- **IODC Registers.** Each peripheral device must implement these registers; see [13] and [11] for a list of the IODC registers required of a GSC+ device.

- **Address/Data Parity.** GSC+ bus requires parity accompany addresses and data; however, LANai ver. 2.3 doesn't generate parity. (LANai ver. 4.0 does.)

- **Host DMA Engine.** In order to make the LANai's EBus-DMA engine adaptable to busses other than SBus, the engine was left incomplete; that is, the engine lacks the control logic that makes it specific to any particular DMA protocol. Instead, the LANai exposes a number of control signals that allow the designer to add logic to control the engine's operation to make it meet the desired DMA protocol.

These requirements make the interface quite complex. Because of this, I decided to design a two-stage interface consisting of two chips: the Slipstream and an FPLD "glue" chip.

2.2 Slipstream

The Slipstream [26] is a custom VLSI chip developed by HP to be the GSC+ bus interface for the HP Fibrechannel adapter card [27]. The Slipstream interfaces the off-card GSC+ bus to an on-card bus called Tachyon System Interconnect, or TSI. TSI is a 5-volt bus, but otherwise it is almost identical to GSC+. The Slipstream relays DIO and DMA transactions between GSC+ bus and TSI. The Slipstream also implements the card's IODC registers. When the card is queried during system initialization, the Slipstream fetches the card's configuration parameters from a pre-programmed on-card FLASH EEPROM chip and returns them to the host processor.

Since the Slipstream was the only chip in existence which supported these features, it was the obvious choice to be the GSC+ bus interface for the Myrinet card. However, it was not ideal; it imposed a number of undesired contraints:

- The Slipstream presents only a 16-kilobyte contiguous address space to the GSC+ bus; *i.e.*, it passes DIO transactions from the GSC+ bus to TSI whose addresses fall within a contiguous 16-kilobyte range. Since the Myrinet card has a 128- or 512-kilobyte local memory, this means most of the memory couldn't be directly mapped into the host processor's address space and thus wouldn't be DIO-accessible.
- As with the GSC+ bus, the Slipstream requires TSI addresses and data to include parity.
- The Slipstream imposes a one-cycle penalty when relaying transactions between the GSC+ bus and TSI.

Completing the GSC+-EBus interface required working around these limitations, and also providing the control logic for the LANai's EBus-DMA engine. Providing these features required adding a second stage to the interface, which is implemented in the FPLD "glue" chip.

2.3 Glue Chip

The glue chip provides the functions that works around the Slipstream's constraints and complete the GSC+-EBus interface. The Berkeley card's glue chip is implemented in an Altera 7256 field-programmable logic device (FPLD) [1]. The Myricom card's glue chip is implemented in AT&T's ORCA FPLD [3]. Both glue chips implement almost exactly the same functions:

- **Local Memory Windowing.** The glue chip provides a windowing function that enables the host processor to access any location in the card's local memory. This windowing function partitions the local memory into 4-kilobyte pages. Three 4-kilobyte page windows are mapped into the host processor's address space, and each window has a page map register associated with it that allows the host processor to select which page is mapped into that window. Figure 2.5 shows how the page windows and page map registers are included in the memory image the Myrinet card presents to the host processor.

As an example, say the host processor wishes to access address 0x5678 in local memory. It first determines the page number and offset within the window as follows:

Page number	= Addr	DIV	0x1000 =	0x5678	DIV	0x1000 =	5
Window offset	= Addr	MOD	0x1000 =	0x5678	MOD	0x1000 =	0x678

Address 0x5678 is in page 5 at offset 0x678. The host sets one of the page map registers (*e.g.*, register 1) to 5. The host can then access address 0x5678 in local memory by using DIO transactions to access offset 0x678 within window 1.

- **Interrupt Mask.** (Myricom card only.) The glue chip provides an interrupt-on-DIO-write function. When enabled, this function issues a wake-up interrupt signal to the LANai whenever the host processor does a DIO write to local memory whose address falls on a specfied alignment boundary. For example, the interrupt mask can be set to issue a wake-up interrupt to the LANai when the host does a DIO write to local memory using an address equal to zero MOD 0x2000.

  This feature was implemented in the Myricom card to support HP's Hamlyn network protocol [6]. The HP-LAM software does not use LANai wake-up interrupts; therefore the interrupt mask registers were not implemented in the Berkeley card.

- **Host DMA.** The glue chip implements control logic in the form of a state machine that directs the LANai's host-DMA engine to perform GSC+-compliant DMA transactions. The GSC+ specification requires DMA transfers to consist of a series of 32-bit-word reads or writes, and that transfers be partitioned into bursts of no more than eight words each.

  The LANai processor core requests a host-DMA transfer by writing to a set of memory-mapped registers. Requesting a DMA transfer causes the following events to occur:

  1. The LANai's host-DMA engine outputs signals indicating the length, alignment, and direction (local-to-main-memory, or main-to-local-memory) of the transfer. The DMA state machine in the glue chip responds by issuing a Bus Request signal to TSI, which the Slipstream relays to the GSC+ bus.
  2. The U2 returns a Bus Grant signal, which the Slipstream relays to TSI and the the DMA state machine. The state machine signals the host-DMA engine to output to EBus the address of the first burst of the DMA transfer.
  3. If this is a local-to-main-memory DMA transfer, the state machine then pulses the host-DMA engine to read from local memory and output to EBus each data word in the burst. If this is a main-to-local-memory transfer, the state machine waits for the data to arrive from main memory; when it does, the state machine pulses the host-DMA engine to accept each word of the burst and write it to local memory. With each pulse, the state machine also signals the host-DMA engine to increment the address counter, decrement the length counter, and include the current data word in the running CRC value.

4. After finishing the burst, the state machine checks if the host-DMA engine's length register has reached zero. If it has, the state machine removes Bus Request, releasing the GSC+ bus. Otherwise, the state machine keeps control of the bus and begins another burst. It repeats this cycle until the entire transfer is completed.

- **EBus Parity Generator.** (Berkeley card only.) The Slipstream's TSI requires that parity accompany address and data, but LANai ver. 2.3's EBus does not include a parity signal. The glue chip generates parity for addresses and data originating from the LANai.

  An EBus parity generator was added to the LANai in ver. 4.0, the version used in the Myricom card. The Myricom card's glue chip therefore did not need to implement this function.

Figure 2.4 shows a block diagram of both cards' glue chips. The memory image that the cards present to the host processor is shown in Figure 2.5.

2.4 Local Memory

The local memory is a high-speed (6-nanosecond access time) static random-access memory (SRAM). The Berkeley card contains 128 kilobytes of SRAM; the Myricom card contains 512 kilobytes. The exact use of the local memory is up to the LCP, but all network messages must pass through it. The local memory also holds the LCP code itself and any other needed data or state. The design of the local memory was taken verbatim from Myricom's SBus-Myrinet card.

2.5 Transceiver

The Berkeley and Myricom cards support different network link signalling protocols, and they therefore differ substantially in the design of their network transceivers. Myricom initially used Partial Emitter-Coupled Logic (PECL) signalling in their network links, but they are replacing it with Low-Voltage Differential Signalling (LVDS).

The Berkeley card uses LANai ver. 2.3 which contains a PECL transmitter and receiver. Ver. 2.3 requires off-chip amplifiers to drive the network link. The Berkeley card uses five AT&T 41MM amplifier chips [4] for this purpose. The design was taken verbatim from the SBus-Myrinet card.

The Myricom card uses LANai ver. 4.0 which does not contain its own network signalling transceiver. Instead, ver. 4.0 is designed to be used with a separate transceiver that implements the signalling protocol. The Myricom card uses a single LVDS transceiver chip custom-designed by Myricom.

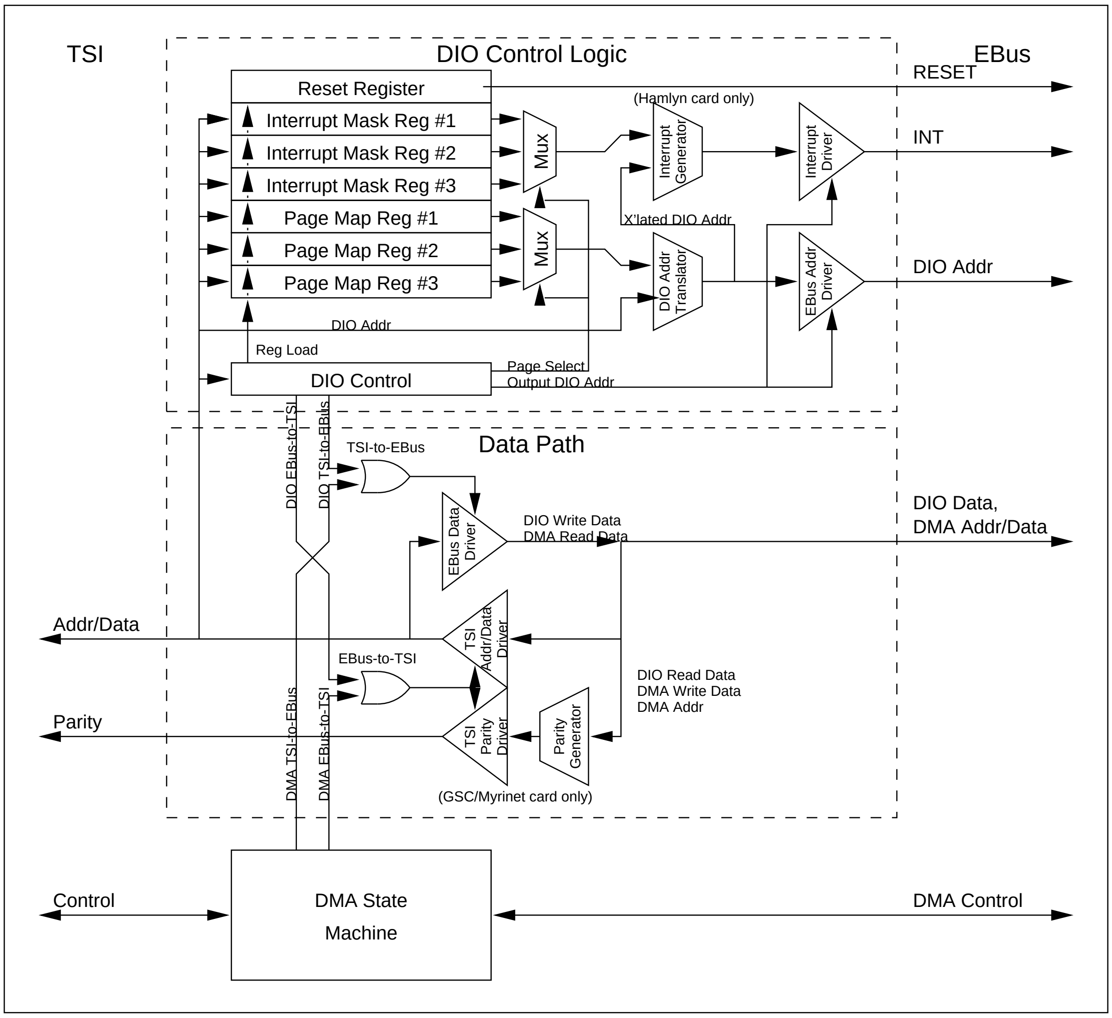

Figure 2.4: **Glue Chip Block Diagram.** The reset register allows the host processor to toggle the LANai's reset signal. The page windows and page map registers form the windowing function that enable DIO access to the entire local memory. The interrupt mask registers (Myricom card only) form the interrupt function that wakes up the LANai when the host processor does aligned DIO writes to the card's local memory. The DMA state machine guides the LANai's host-DMA engine in performing DMA transactions that are GSC+-compliant. The EBus parity generator (Berkeley card only) generates parity for LANai-sourced EBus addresses and data.

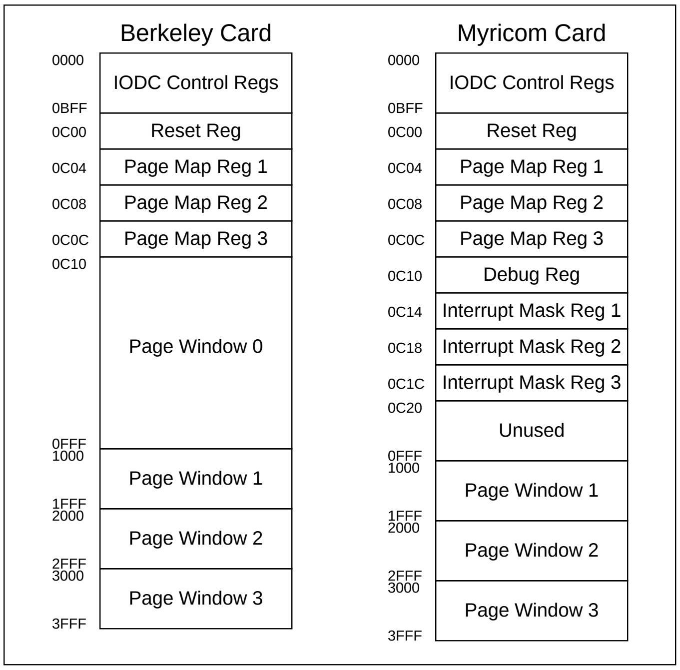

Figure 2.5: **Myrinet Interface Card Memory Image.** Together the Slipstream and the glue chip create this memory image in the host processor's address space. The PA-RISC I/O specification requires IODC registers in all peripherals for identification and initialization. The reset register enables the host processor to reset the LANai. The three windows and three page map registers make up the windowing function that enables the host processor to access any location in the card's local memory. The Berkeley card also includes a Page window 0 which is hard-wired to the upper portion of local memory page 0. The three interrupt mask registers (Myricom card only) cause a LANai wake-up interrupt to be triggered when the host processor does a DIO-write to local memory whose address falls on the specfied alignment boundary.

2.6 Discussion

A number of alternatives were considered during the design of the Myrinet interface card:

- Because of the Slipstream's limitations, I originally considered not using it at all, instead implementing the GSC+-EBus interface in a set of FPLDs. Because of the complexity of the interface (sec. 2.1), I decided this was too risky, that using the Slipstream was the safest way to achieve functionality.
- It would have been possible to design a usable card without implementing the local memory windowing scheme (sec. 2.3). The host processor could have been constrained to accessing only the first 16 kilobytes of the local memory. The HP-LAM data structures that the host processor must access would have been mapped into this space. The rest of local memory would be accessed only through the host-DMA engine. Despite the greater complexity, I decided to implement the register windowing function anyway to give the card greater flexibility.
- In designing the Berkeley card, I considered a number of FPLD vendors before selecting Altera: various 22V10 PAL suppliers, Cypress Semiconductor, and others. I chose Altera because I felt they offered the best package of FPLD sizes, advertized performance, and development tools.

My biggest regret in this project is not being able to complete the development of the Berkeley card. The card never did achieve functionality. At the time we suspended the development effort, the card had been fabricated and assembled, DIO access to the Slipstream and the glue chip was working correctly, and I had completed programming and debugging the glue chip's DIO control logic and data path. I was in the process of debugging the glue chip-LANai interface when we decided to suspend the development effort and instead use the Myricom card for HP-LAM software development and testing, since by this time the Myricom card had been available for several months.

There are several reasons why development of the Berkeley card took longer than expected. The biggest mistake I made in developing the card was neglecting the EBus parity generator until well into the design. In early versions of the design, the TSI and EBus address/data lines were tied directly together. I expected to use an off-the-shelf parity generator chip to generate parity. Unfortunately, this approach assumed that a commodity chip could generate parity within a single TSI/EBus clock cycle (under 20 nanoseconds). It turned out that no such chip existed that was fast enough. This forced me to redesign the card with a data latch in the glue chip to give the parity generator an extra cycle to generate parity. No longer being able to tie the TSI and EBus lines directly together greatly increased the complexity of the glue chip, causing the development to be delayed by months.

Myricom was aware that I had run into the parity generation problem when they began designing their card. Myricom solved this problem by simply adding a parity generator to the LANai in ver. 4.0. (After all, they produce the LANai.) At Berkeley, however, we were too far along with the design of our card to change to using the LANai ver. 4.0. In retrospect, we should have worked more closely with Myricom to solve this problem.

Chapter 3

Software Design

HP-LAM is the software implementation of Generic Active Messages (GAM) on the HP 9000/J200 workstation, the HP-UX operating system ver. 10.00 [14], and the Myrinet interface card. The acronym *HP-LAM* stands for *Hewlett-Packard LANai Active Messages.* It is so named because it is largely derived from Sun-LAM, the GAM implementation on the Sun Microsystems SparcStation and Ultra workstations and Myrinet [19].

HP-LAM consists of several components. The components were derived from various sources: from Sun-LAM, from HP Hamlyn, and from Myricom. The task of implementing HP-LAM was primarily one of importing these pieces of software, modifying them as necessary, and integrating them. Figure 3.1 shows the HP-LAM software architecture, and the components are listed below:

- **The GAM library** includes all of the functions that each node of the parallel program calls to communicate with the other nodes. The GAM library is linked into the program and thus resides in the user's virtual address space.

- **The Myrinet library** includes the low-level functions that the GAM library calls to communicate with the Myrinet network interface. This library is also linked into the parallel program and resides in user-space. The GAM and Myrinet library functions are in separate libraries to separate functions used in other GAM implementations from functions specific to the Myrinet network interface. This allows the same GAM library source code to be used for multiple platforms.

- **The `iomap` device driver** is a standard HP-UX device driver that the GAM and Myrinet libraries use to establish access to the Myrinet interface card. `iomap` is the generic mechanism used to access all peripheral devices. An application or another device driver may call it to map a device's memory image into its address space. As with all HP-UX device drivers, `iomap` is linked into the kernel and resides in the kernel's virtual address space.

- **The LAM device driver** is used by the Myrinet library to manage the DMA copy block. The DMA copy block is an area reserved in main memory from which the Myrinet card fetches and stores bulk data using its host-DMA engine. The LAM

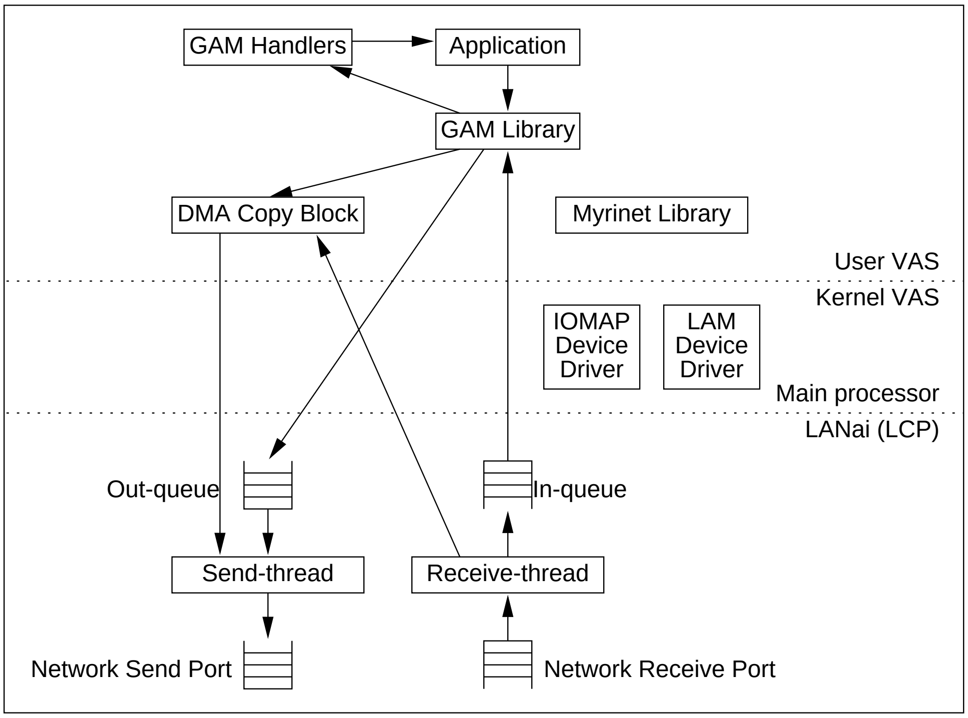

Figure 3.1: **HP-LAM Software Architecture.** The program calls GAM library functions to send and receive messages. To send a message, the GAM library copies bulk data (if any) to the DMA copy block, then enqueues the message in the out-queue. The LCP dequeues the message, grabs the bulk data, and writes it into the network. At the receiving node, the LCP reads the message out of the network, copies the data to the DMA copy block, and enqueues the message in the in-queue. The GAM library dequeues the message and grabs the data from the copy block. The Myrinet library, the `iomap` device driver, and the LAM device driver are used to set up the mechanism.

device driver provides three functions: (1) it "wires down" the copy block; *i.e.*, it disables the HP-UX virtual memory system from paging the block out of main memory; (2) it maps the block into the GSC+ bus' address space in order to make it accessible to the Myrinet card; and (3) it makes the program non-swappable; *i.e.*, it prevents the HP-UX scheduler from swapping in another user process while the program is active. The LAM device driver also resides in kernel-space.

- **The LANai Control Program (LCP)** runs in the LANai processor core. It consists of two separate, independent threads. The *send-thread* fetches bulk data from the DMA copy block and injects outgoing messages into the network. The *receive-thread* reads incoming messages from the network and stores bulk data to the copy block.

The following paragraphs give an overview of HP-LAM's operation by following an active message sent between two nodes.

Early in the execution of the parallel program, each node calls `am_enable()` to initialize its local Myrinet interface card. `am_enable()` reads local configuration information, establishes unshared access to the card, creates the DMA copy block, downloads the LCP to the card's local memory, and sets up data structures.

To send a message to another node, the source node calls a GAM library function such as `am_request()`. The GAM function formats the request on the node's host processor and copies it to the out-queue in the card's local memory. For bulk transfers, the GAM function also copies the transfer data into the DMA copy block in main memory to make it accessible to the card.

The LCP executing in the LANai processor core consists of two threads: a send-thread for handling outgoing messages, and a receive-thread for incoming messages. The send-thread continuously polls the out-queue. When it finds a message in the queue ready to be sent, it first checks if the message transfers bulk data. If it does, the send-thread copies the data from the DMA copy block in main memory to the bulk send buffer in local memory using the LANai's EBus-DMA engine. It then copies into the network the route to the destination node from the routing table in local memory, followed by the message from the out-queue. For a bulk transfer the send-thread then uses the send-DMA engine to copy into the network the transfer data from the bulk send buffer. Finally, it tells the link interface that the message is completed, which causes the interface to append a CRC byte to the message to mark its end.

At the destination node, the receive-thread continuously polls the network interface for incoming messages. When one arrives, the receive-thread first copies the message header into the in-queue, but for the momemt it leaves the queue entry marked as invalid. Next it then checks if the message is a bulk transfer. If it is, the receive-thread uses the receive-DMA engine to copy the data from the network into the bulk receive buffer in local memory, then the EBus-DMA engine to copy the data to the DMA copy block in main memory. Finally, the receive-thread marks the message in the in-queue as valid.

Each node in the parallel program is required to regularly poll the in-queue in the card's local memory for received messages. Calling any GAM function implicitily does this. If a node would otherwise not call any GAM functions for an extended time, it must call the function `am_poll()` to explicitely check the in-queue. When a received message is found,

Component	Origin	Adaptation to HP-LAM
GAM Library	Sun-LAM	Modified access to card
Myrinet Library	HP-Myrinet Library	Modified access to card and device driver
`iomap` Device Driver	HP-UX	Unchanged
LAM Device Driver	HP Hamlyn	Used only selected parts
LANai Control Program (LCP)	Sun-LAM	Aligned in- & out-queues

Table 3.1: **HP-LAM Software Components.** HP-LAM consists of several components, each component derived from an outside source.

the GAM library calls the handler function identified by that message. If this is a request, the handler is expected to return a reply to the source; if it doesn't, the AM library will return a reply itself. If this is a reply, the message is considered delivered once the handler finishes executing.

Table 3.1 lists each component in the HP-LAM software and identifies its origin. The rest of this chapter describes each component in more detail.

3.1 GAM Library

The GAM library includes the functions that the nodes in a parallel program use to communicate with each other. Nodes send and receive messages solely by calling GAM library functions. A list of GAM functions appears in [9].

The parallel program must provide the handlers used by the GAM functions. The GAM library and the handlers are linked into the program at compile-time; thus the GAM functions and handlers are executed as part of the program image in user-space.

An active message consists of two parts, a **request** and a **reply**. In HP-LAM requests and replies come in three sizes, **short**, **medium**, and **bulk**. Short messages contain the source and destination node numbers, a pointer to the handler to be called at the destination, and the handler arguments. Bulk messages contain in addition two pointers, one pointing to data on the source node to be transferred to the destination, the other pointing to where to deposit the data on the destination node. Medium messages are similar to bulk messages except that they contain only a single pointer that points to the data on the source node to be transferred. A medium message deposits the data in the destination node's DMA copy block and gives the handler a pointer to the data. The handler is expected to read the data from there.

HP-LAM uses credit-based flow control to prevent buffer overflow: each node in the parallel program may have no more than n messages outstanding at a time. Each node begins with n credits. When a source node issues a request, it grabs a credit; when a reply is received, the credit is released. If all of the credits on a node are in use when it issues a request, the request must wait until a credit becomes available. Thus the credit count on a source node is modified only when it *sends a request* or *receives a reply*. Requests are received and replies are sent by request handlers on the destination node, and they do not modify the credit count.

The GAM library functions pass requests and replies to the Myrinet card by enqueuing them in the out-queue in the card's local memory. To send a bulk message, the data is first copied to the DMA copy block. The DMA copy block is divided into send- and receive-partitions, and these partitions are divided into 4-kilobyte pages. Each page in the send-partition is associated with one entry in the out-queue, and each page in the receive-partition is associated with one entry in the in-queue. In a bulk send the GAM function determines which is the next available entry in the out-queue, and it copies the bulk data to the corresponding page in the send-partition. To send bulk data greater than 4 kilobytes, the GAM function divides the data into 4-kilobyte segments and sends each segment using a separate request-reply pair.

The following procedure is used to send a message:

```
IF (medium or bulk message) AND (sizeof(transfer-data) > maximum allowed) THEN BEGIN
   Break the transfer-data into several smaller blocks.
   Transfer each block using a separate message, the last message calling the handler.
END IF

IF (request) THEN BEGIN
   IF (no credits are currently available) THEN BEGIN
      Wait until a credit becomes available; if none becomes available
         after a specified period, give up and return an error.
   END IF
   Decrement the credit count; i.e., grab a credit for this message.
END IF

IF (medium or bulk message) THEN BEGIN
   Copy the transfer-data to the DMA copy block.
   (Each out-queue entry has an address associated with it in the copy block.
   The LCP will grab the data from this address after the message is enqueued.)
END IF

Enqueue the source and destination node numbers, the handler pointer, and the
   handler arguments in the out-queue in the interface's local memory.
```

To receive messages, each node in the parallel program must regularly call any GAM library function. All GAM functions implicitely poll the in-queue. If a node would not otherwise call any GAM function for an extended time, it must call the function `am_poll()` to explicitely check the in-queue. When any received messages are found, they are delivered as follows:

```
BEGIN LOOP
   Check the in-queue in the interface's local memory for any received messages.
   IF (no received messages) THEN RETURN.

   Dequeue a message from the in-queue.

   IF (message has a CRC error) THEN BEGIN
      Report the CRC error.
      Discard the message.
   END
   ELSE BEGIN
      IF (bulk message) THEN BEGIN
         Copy the transfer-data from the DMA copy block to
            the destination address indicated in the message.
```

```
      (As with the out-queue, each entry in the in-queue has an address associated
      with it in the copy block—the LCP will have already put the data there.)
    END IF

    Call the handler.
    (If this is a medium message, the handler is expected to read the transfer-data directly out
    of the copy block. The message is considered delivered once the handler finishes executing.)

    IF (request) THEN BEGIN
      IF (request handler didn't return a reply) THEN
        Send a reply to the source.
      END IF
    END
    ELSE IF (reply) THEN BEGIN
      Increment the credit count; i.e., release the credit associated with this message.
    END IF
  END IF
END LOOP
```

Note that this procedure drains *all* of the messages in the in-queue before returning.

In a bulk message the bulk data is deposited in the DMA copy block receive-partition page that is associated with the in-queue entry that holds the message. The GAM function copies the bulk data from this page to the address indicated in the message before it calls the handler. In a medium message the GAM function passes a pointer to the receive-partition page to the handler, and the handler is expected to read the data from there.

The GAM library used in HP-LAM was derived from the library used in Sun-LAM. In adapting the Sun-LAM GAM library to HP-LAM, the library needed to be substantially rewritten to accomodate the Myrinet interface's local memory windowing mechanism (sec. 2.3). Since there are three Page Map registers, each register is dedicated to a particular type of access:

- Page window 1 is used to access the in-queue. Since the GAM library delivers messages serially, only one in-queue entry is accessed at a time. After each message is read from the in-queue, the in-queue pointer is incremented and Page Map register 1 is updated.
- Page window 2 is used to access the out-queue in the same manner as the in-queue. After each message is written to the out-queue, the out-queue pointer is incremented and Page Map register 2 is updated.
- Page window 3 is used for all other Myrinet card accesses, none of which are on the critical path of sending and receiving messages. Page Map register 3 is updated with every read and write to the local memory.

3.2 Myrinet Library

The Myrinet library contains low-level functions for accessing and controlling the Myrinet interface card. These functions allow the user to reset and initialize the card, read and write to the Slipstream and glue chip registers, access local memory, and enable and disable interrupts (not used by HP-LAM). The library also provides functions for creating and managing the DMA copy block. The Myrinet library is linked into the parallel program at compile-time and is thus executed in user-space.

It isn't necessary to use the Myrinet library for all accesses to the interface card. Once access is first established, the card can be accessed directly using memory-mapped I/O. The GAM library calls the Myrinet library to initialize the card and download the LCP. The Myrinet library isn't used, however, for sending and receiving messages.

The Myrinet library was derived from the HP library that Myricom provides with their interface card. In adapting the library for HP-LAM, I changed it substantially in the way it creates and manages the DMA copy block. The original library allocated the copy block by calling `mmap()` on the `/dev/kmem` device special file. This is the method used in Sun-LAM; it is a simple, elegant hack that allows the user to access a memory block in kernel virtual address space. HP-UX, however, doesn't allow `mmap()` to be called on device special files. Instead, it was necessary for me to create a device driver (sec. 3.4) to manage the copy block. The Myrinet library now calls `brk()` to create the copy block in user-space and then calls the LAM device driver to wire down the copy block and map a GSC+ address range to it.

3.3 `iomap` Device Driver

The `iomap` device driver is a standard HP-UX driver that the GAM and Myrinet libraries use to establish access the Myrinet interface card. `iomap` is the generic mechanism used to access all peripheral devices; it maps the device's memory image into the kernel or an application's virtual address space.

The Myrinet library establishes access to the interface card by calling the `open()` and `ioctl()` system functions. These calls are routed to the `iomap` driver; it grants non-shared access to the card and maps the card's memory image (see Figure 2.5) into the program's address space.

The `iomap` driver used by HP-LAM is the standard HP-UX ver. 10.00 driver; it required no modification.

3.4 LAM Device Driver

The sole purpose of the LAM device driver is to manage the DMA copy block. In order to make the copy block accessible to the interface card's host-DMA engine, it must be "wired down;" that is, the block must be made non-pageable so that physical memory will always be assigned to the block for the card's DMA engine to access. The block must also be mapped into GSC+ address space to make it accessible by the card. Finally, the program must not be swapped out of main memory. In HP-UX, these are privileged operations available only to the kernel. The LAM device driver was created to make them available to HP-LAM.

After the Myrinet library creates the DMA copy block in user-space, it opens a connection to the LAM driver by calling `open()` on a device special file linked to the driver. The library then passes a pointer to the block to the driver using `ioctl()`.

The driver first checks the validity of the block: that the block is contained in the program's data segment, that the block is read/write-accessible by the program, and that the program hasn't already opened another block. Once satisfied, the driver makes the program non-swappable. Next, the driver touches each page in the copy block to associate it with a page of physical memory, and it then calls the kernel function `mlockpreg()` to wire down the pages (*i.e.*, make them non-pageable). Finally, the driver maps the block into GSC+ address space by adding entries to the U2 bus adapter's [29] page tables (these are a subset of the system's page tables).

After completing this there are four addresses for the DMA copy block: in user virtual address space, in kernel virtual address space, the physical memory address, and in GSC+ address space. The driver returns the GSC+ address to the Myrinet library for it to pass on to the LCP.

The LAM device driver wouldn't be needed at all if host-LANai DMA weren't used. Early versions of HP-LAM didn't use DMA, but it was added to improve the performance of bulk transfers.

I wrote the LAM device driver myrself using similar functions in HP Labs' Hamlyn [6] device driver as reference.

3.5 LANai Control Program (LCP)

The LCP runs in the LANai processor core. It marshalls the LANai's resources to send and receive messages. In essence the LCP makes the LANai acts as two queues, one outbound to the network, the other inbound.

The LCP consists of two separate, independently running threads: the **send-thread** and the **receive-thread**. Both threads essentially act as producer-consumer queues for messages passing between the GAM library and the network. The two threads are time-shared in the LANai processor core, and the receive-thread is given higher priority to ensure that the network drains properly and remains deadlock-free.

The send-thread executes the following algorithm:

```
BEGIN LOOP
  IF (message is in the out-queue) THEN BEGIN
    Dequeue the message from the out-queue.
    Fetch the destination route from the routing table in local memory.
    Write the route and the message to the network link using the send-DMA engine.

    IF (bulk or medium message) THEN BEGIN
      IF (this message's transfer data hasn't already been prefetched) THEN
        Copy the transfer data from the DMA copy block to
          the bulk send buffer using the host-DMA engine.

      Check the next message in the out-queue.
      IF (bulk or medium message)
        Prefetch the next message's transfer data from
          the DMA copy block using the host-DMA engine.

      Copy the transfer data to the network link using the send-DMA engine.
    END IF

    Mark the end-of-message in order to add a CRC byte to the message.
  END IF
```

```
END LOOP
```

Note that the send-thread tries to prefetch the next message's transfer data in order to increase throughput.

The receive-thread executes a similar algorithm:

```
BEGIN LOOP
  IF (incoming message pending) THEN BEGIN
    IF (in-queue is full) THEN
      Drop the message.
    ELSE BEGIN
      Use the receive-DMA engine to copy the message header
        in from the network link to the in-queue.

      IF (short message) THEN BEGIN
        IF (CRC error) THEN
          Drop the message.
        ELSE
          Mark the message in the in-queue as received.
      END
      ELSE IF (bulk or medium message) THEN BEGIN
        IF (transfer data would overflow the bulk receive buffer in local memory) THEN
          Drop the message.
        ELSE BEGIN
          Use the receive-DMA engine to read the transfer data
            in from the network link to the bulk receive buffer.

          IF (CRC error) THEN
            Drop the message and the transfer data.
          ELSE BEGIN
            Use the host-DMA engine to write the transfer data
              from the bulk receive buffer to the DMA copy block.
            Mark the message header in the in-queue as received.
          END IF
        END IF
      END IF
    END IF
  END IF
END LOOP
```

Note that the receive-thread will drop messages if either the in-queue or the bulk receive buffer in local memory are full. The GAM library implements a credit-based flow control mechanism that should prevent this from ever happening (sec. 3.1), but it was still included to make sure the network won't deadlock.

The HP-LAM LCP was taken from the Sun-LAM LCP for LANai ver. 4.0 (L4LCP). The only change needed was to accomodate the new windowing scheme (sec. 3.1). In order to allow every word in a queue entry to be accessed without having to set the Page Map register for each one, the queues were aligned so that individual entries don't straddle page boundaries. This allows the Page Map register to be set once to access the entire queue entry.

3.6 Discussion

Despite GAM's complexity and interdependence with other software layers, it is mature enough that porting it to a new platform was relatively easy. I had HP-LAM without DMA support working on the 9000/J200 after a few weeks' effort. Supporting DMA transfers between the Myrinet card and main memory, however, was a major challenge. The HP-UX operating system presented several obstacles:

- HP-UX doesn't provide any user-level functions for wiring down memory pages or for mapping pages into GSC+-bus address space. I had to write my own device driver to do it. Moreover, there is no assurance that the method I used will be upward-compatible; HP-LAM is guaranteed to work only with HP-UX ver. 10.00.

- HP-UX doesn't support the dynamic installation of device drivers. After being recompiled, a driver must be linked into the kernel and the system rebooted. This turned out to be quite a nuisance during debug.

- The `mmap()` function can't be called on device special files such as `/dev/kmem` (see sec. 3.2).

I present these to the HP-UX operating system development group as suggestions for future enhancements.

During development I tried several methods of accomodating the local memory windowing mechanism. The first working version used the Myrinet library functions `lanai_read_word()` and `lanai_write_word()` for all accesses to local memory. Each call to these functions sets Page Map register 3 to the desired page of local meory and accesses the desired offset through Page window 3. The overhead of setting the Page Map register with each access caused the performance of this version to be poor, which led me to develop the present method (sec. 3.1).

Chapter 4

Performance

This section describes the functionality and performance tests done on HP-LAM. It gives interpretations on the results, and it compares the performance of HP-LAM to that of other GAM implementations.

There are three groups of tests: the GAM basic benchmarks, the LogP tests, and several split-C applications. Every GAM implementation mentioned in this report, including HP-LAM, passes every test described.

4.1 GAM Basic Benchmarks

The GAM basic benchmarks verify that the GAM software is functioning correctly. They are the first software tests applied to any GAM implementation. The benchmarks also yield the average round-trip time (RTT) for small messages and the average bandwidth (BW) for bulk messages. The benchmark suite contains the following tests:

- The **ping_pong** test sends a number of messages between two nodes and measures the average RTT. The first node sends an `am_request()` to the second node; the second node's request handler sends an `am_reply()` back to the first node. This is repeated n times; average RTT = total_time / n.

- The **ping_bulk** and **thru_bulk** tests measure the average BW between two nodes. In **ping_bulk**, the first node calls `am_store()` to do a bulk transfer to the second node. The second node's request handler then calls `am_reply_xfer()` to do a bulk transfer back to the first node. This is repeated n times; average BW = $2n$ * packet_size / total_time. **thru_bulk** works slightly differently: the source node calls `am_store()` to the destination node n times without waiting for each transfer to complete. **thru_bulk** exposes the network BW by filling the network to its capacity, effectively causing each `am_store()` to block until the network can absorb another bulk message. Average BW = n * packet_size / total_time.

- **test_xfer** and **test_med** are torture tests for the GAM software and network hardware. **test_xfer** performs the following sequence of tests:

	Write Time, nsec	Read Time, nsec
Main Memory	20	20
Slipstream	133	595
Glue Chip	183	627
Local Memory	183	691

Table 4.1: **Myrinet Interface Card Access Times.** These are average times for the host processor to perform reads and writes to various components on the card. The time for the host processor to access main memory is given for comparison.

1. **1-to-1.** Two nodes perform a number of `am_get()`s and `am_store()`s between themselves. The transferred data is checked for correctness. Surrounding memory regions are checked that no data was written outside of the destination blocks.
2. **All-to-1.** All nodes perform a number of `am_get()`s and `am_store()`s to node 0. Again, all of the transferred data is checked for correctness.
3. **All-to-all.** All of the nodes perform a number of `am_get()`s and `am_store()`s to each other. The transferred data is checked as before.

test_med is identical to **test_xfer** except that it uses medium instead of bulk messages.

These tests show that the RTT for HP-LAM is **28.1 μsec**, and the BW is **24.0 MB/sec**. One may find these disappointing, given that the nominal BW for the GSC+ bus is 160 MB/sec and for Myrinet is 80 MB/sec.

A study of the RTT is revealing. Table 4.1 shows the times to read and write various components on the Myrinet interface card. The values show that the time for performing the simplest access to the card, a write to a Slipstream register, is 113 nsec greater than the time to access main memory. This is clearly a result of the write transaction having to cross from the system bus to the GSC+ bus. More surprising is the time for the host processor to read from the card. The fastest read, a read from a Slipstream register, takes 575 nsec longer than a read from main memory. This is caused by the double-penalty of the read transaction crossing from the system bus to the GSC+ bus and the data having to cross back.

From this it is easy to see how the RTT accumulates so quickly. Each in-queue and out-queue entry contains eight words. Accessing a queue entry requires one write to set the Page Map register followed by eight reads or writes to the card's local memory. The round time for an active messages entails enqueuing the request at the source, dequeuing the request and enqueuing the reply at the destination, and dequeuing the reply at the source. The time for these accesses alone adds up to 14.7 μsec. Add in the time spent formatting the messages, polling the queues, and executing the handlers, it isn't surprising that the RTT is 28.1 μsec.

The low BW is also a result of the overhead of crossing between busses. Recall that a DMA transfer consists of a series of eight-word bursts. In a host-to-local-memory

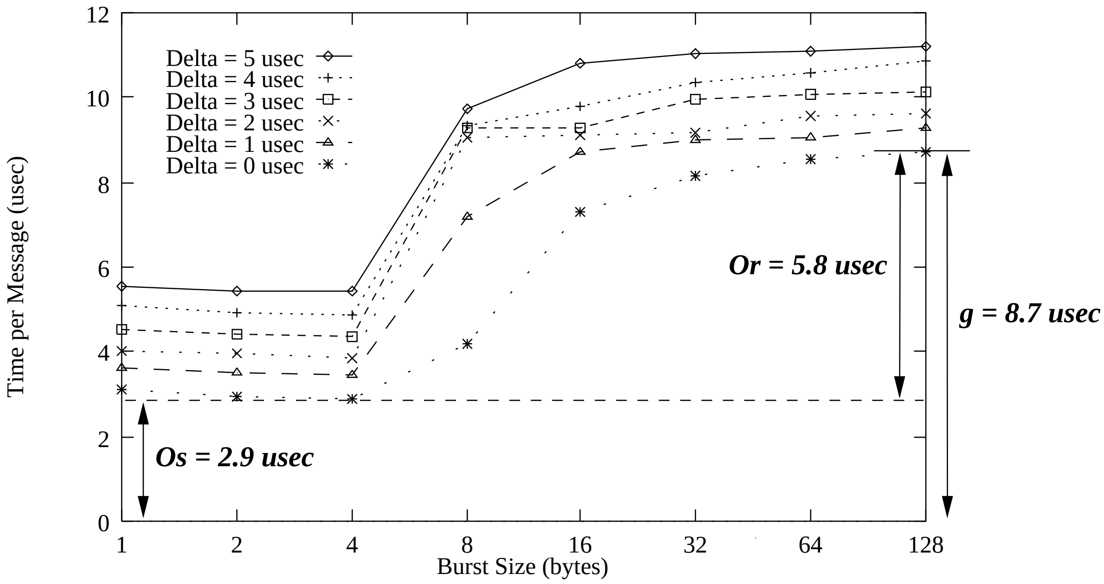

Figure 4.1: **HP-LAM LogP Performance Plots.**

DMA transfer, the read request from the LANai's host-DMA engine must cross the card's EBus and TSI bus and the system's GSC+ bus and system bus to reach memory. Data must return across this path before being stored in the local memory. Even with each burst moving eight words of data, it is apparent that a high percentage of bus cycles will be spent passing transactions and waiting for replies, not actually moving data.

4.2 LogP

The LogP network performance model [8] breaks down the RTT for small messages into four components. Send overhead o_s and receive overhead o_r are the amount of processing time the source and destination host processors devote to sending and receiving each message, respectively. Latency L is the time a message spends traversing the network without occupying either the source or destination processor. The fourth parameter, the gap g, indicates the capacity of the network and is inversely proportional to the bandwidth. g indicates the minimum amount of time between the sending of adjacent messages. The LogP parameters are especially useful for comparing the performance of analogous components in different GAM implementations. For example, if the send overhead for a particular implementation were found to be especially low, it would encourage investigation into how such performance was achieved.

Figure 4.1 shows the LogP signature plot for HP-LAM. From these we derive the LogP parameters which are given in Table 4.2. Table 4.2 also compares HP-LAM's RTT, BW, and LogP parameters to those of four other GAM platforms: Sun-LAM on the SparcStation-20 and the Ultra-1, the Intel Paragon, and the Meiko CS-2 [10].

Figure 4.1 illustrates how the LogP parameters are extracted from the plot. The

	HP-LAM	Sun-LAM on SS-20	Sun-LAM on Ultra-1	Intel Paragon	Meiko CS-2
Round-trip time (RTT, μsec)	**28.1**	31.0	25.0	20.0	20.9
Bandwidth (BW, MB/sec)	**24**	20	38	145	45
Latency (L, μsec)	**5.3**	11.2	5.5	6.5	7.5
Send overhead (o_s, μsec)	**2.9**	2.0	2.0	1.4	1.7
Receive overhead (o_r, μsec)	**5.8**	2.6	5.0	2.2	1.6
Gap (g, μsec)	**8.7**	12.4	7.0	7.6	13.6

Table 4.2: **RTT, BW, and LogP Parameters for Five GAM Platforms.**

send overhead o_s is the time between sending adjacent minimum-sized messages, as is seen on the left side of the plot. The gap g is the time between sending adjacent maximum-size messages, as seen on the right side of the plot. The receive overhead o_r is found by adding an interval called delta-time between the sending of each message until the gap g begins increasing. In this plot the gap begins increasing with the addition of *any* delta-time, which indicates the gap consists *entirely* of send and receive overhead as shown. The latency L is calculated from the other parameters. The time for a one-way trip RTT/2 is made up of o_s, o_r, and L, L: $o_s + o_r + L = \text{RTT}/2$. (A complete explanation of how LogP parameters are obtained is given in [10].)

The comparison of HP-LAM's and Sun-LAM's performance is significant, since the platforms are so similar. Send and receive overheads are higher for HP-LAM than for either version of Sun-LAM. This is most likely because transactions between the host processor and the Myrinet card must pass through more interfaces in the 9000/J200 than in either Sun platform. HP-LAM's latency is half that of Sun-LAM's on the SparcStation-20 and about equal to that on the Ultra-1. This is most likely due to the fact that the 9000/J200's LANai is clocked at 40 MHz, the SparcStation-20's at 25 MHz, and the Ultra-1's at 37.5 MHz.

4.3 Split-C Applications

Split-C [7] is a parallel extension of the C programming language developed at U. C. Berkeley. Split-C creates the image of a global address space in an MPP or a cluster of workstations. It provides global access primitives and simple parallel storage layout declarations. Split-C uses message-passing to create this global image: each node in the cluster implements a fraction of the global address space in its local memory, and a node accesses global addresses by passing messages to the nodes that own each address.

Split-C uses GAM as its message-passing layer, which means it can be used on any platform that supports GAM. Running Split-C programs on a GAM platform is a major proof of the implementation's correctness and performance.

Table 4.3 shows the performance values obtained running the Split-C application `radix sort` on three platforms: the HP 9000/J200 and HP-LAM, and Sun SparcStation-20

			HP-LAM	Sun-LAM on SS-20	Sun-LAM on Ultra-1
Pass 1	Fill	(sec)	**0.14**	0.56	0.13
	Wave-scan	(sec)	**1.04**	0.88	0.95
	Coalesce	(sec)	**9.84**	8.99	7.78
Pass 2	Fill	(sec)	**0.12**	0.25	0.09
	Wave-scan	(sec)	**1.03**	0.80	0.93
	Coalesce	(sec)	**0.65**	1.18	0.27
TOTAL		(sec)	**12.82**	12.66	10.14

Table 4.3: **Performance of `radix sort` on HP-LAM and Sun-LAM.** Radix sort is run in two passes, each pass consisting of a compute-intensive phase, *fill*, and two communication-intensive phases, *wave-scan* and *coalesce*.

and Sun-LAM. Radix sort is run in two passes. Each pass consists of one compute-intensive phase, *fill*, and two communication-intensive phases, *wave-scan* and *coalesce*.

Table 4.3 shows that HP-LAM's performance of the compute-intensive *fill* phase is between 2 and 4 times faster than Sun-LAM's on the SparcStation-20, and it is comparable to Sun-LAM's performance on the Ultra-1. This is not surprising given that the 9000/J200 uses a 120-MHz PA-7200 processor, the SparcStation-20 a 50-MHz SuperSparc processor, and the Ultra-1 a 167-MHz UltraSparc processor. What is surprising is that the performance of the communication-intensive *wave-scan* and *coalesce* phases is actually *slower* for the 9000/J200 than for either Sun workstation. This is no doubt due in part to the 9000/J200's high overhead of transferring data between the host processor and the Myrinet card. However, this result was unexpected and deserves further investigation.

4.4 Discussion

In this chapter we have seen that the Myrinet interface card and HP-LAM work correctly and achieve performance comparable to that of other GAM implementations. We have also seen, however, that the performance is less than it could be. It is clear that the 9000/J200's I/O architecture limits a peripheral's performance, and that the multi-stage interface design of the Myrinet interface card limits performance further. The GSC+ bus advertizes potential bandwidth of 160 MB/sec, the Myrinet links 80 MB/sec, yet HP-LAM is able to achieve 24 MB/sec. In order for a network interface to begin approaching these performance values, a way must be found to remove these layers of inefficiency and integrate the network interface closer to the host processor.

Chapter 5

Summary

In this report I described the design of the Myrinet interface card hardware and the HP-LAM software. I measured the performance of HP-LAM in terms of LogP parameters and in terms of execution time for a Split-C application, and I compared these performance values to those of other GAM implementations. I showed how HP-LAM's performance is comparable to those of the other implementations, but it is nonetheless limited by inefficiencies in transferring data over the peripheral bus and through the card's multi-layer bus interface.

I cited the difficulties encountered in this project because of the challenges in interfacing HP's GSC+ bus to Myricom's EBus. Both are proprietary busses, and interfacing them turned out to be a major challenge that required designing a cumbersome two-layer translator. I also cited the challenges caused by features lacking from the HP-UX operating system that I feel it should include.

Finally, I pointed to the directions that future work in this area should take: that in order for high-speed networks to begin realizing their potential performance, the network interface must be more tightly integrated into the system architecture to remove the current layers of inefficiency.

Bibliography

[1] Altera Corp. "MAX 7000 Programmable Logic Device Family." Aug. 1994.

[2] T. Anderson, D. Culler, D. Patterson, *et. al.* "A Case for NOW (Networks of Workstations)." *IEEE Micro*, Feb. 1995.

[3] AT&T ORCA FPGA.

[4] AT&T 41MM Transceiver Chip.

[5] A. Basu, V. Buch, W. Vogels, and T. von Eicken. "U-Net: A User-Level Network Interface for Parallel and Distributed Computing." In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, CO, Dec. 1995.

[6] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. "An Implementation of the Hamlyn Sender-managed Interface Architecture." In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI '96)*, Seattle, WA, Oct. 1996.

[7] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. "Parallel Programming in Split-C." In *Supercomputing '93*, Portland, OR, Nov. 1993.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. "LogP: Towards a Realistic Model of Parallel Computation." In *Proceedings of the Fourth ACM Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993.

[9] D. Culler, K. Keeton, C. Krumbein, L. Liu, A. Mainwaring, R. Martin, S. Rodrigues, K. Wright, and C. Yoshikawa. "The Generic Active Message Interface Specification," ver. 1.1. Computer Science Division, University of California, Berkeley, Feb. 1995. *http://now.cs.berkeley.edu/Papers/Papers/gam_spec.ps*

[10] D. Culler, L. Liu, R. Martin, and C. Yoshikawa. "Accessing Fast Network Interfaces." *IEEE Micro*, Feb. 1996.

[11] "GSC-2X Specification," rev. 8.4. Hewlett-Packard Company, Palo Alto, CA, Aug. 1995. **HP Confidential**—this document cannot be distributed outside of Hewlett-Packard with appropriate non-disclosure agreements.

[12] "HP Technical Computing—HP 9000 J-Class Family." Hewlett-Packard Company, Palo Alto, CA. *http://www.hp.com/wsg/products/j_cls.html* OTHER SOURCES?

[13] "HP Technical Computing—PA-7200 PA-RISC Processor." Hewlett-Packard Company, Palo Alto, CA. *http://www.hp.com/wsg/strategies/pa7200.html* OTHER SOURCES?

[14] "HP-UX Version 10.10 Operating System." Hewlett-Packard Company, Palo Alto, CA. *http://www-dmo.external.hp.com/gsy/hpux1010.html* OTHER SOURCES?

[15] C. Krumbein and Y. Cho. "GSC/Myrinet Card Specification," ver. 1.1. Computer Science Division, University of California, Berkeley, Aug. 1995. *http://www.cs.berkeley.edu/ ced/hp_lam/gsc_myrinet.1.1b.ps*

[16] A. Mainwaring and D. Culler. "Active Message Application Programming Interface and Communication Subsystem Organization." University of California, Berkeley, Computer Science Division, Dec. 1995. *http://www.cs.berkeley.edu/ alanm/Papers/trdraft.ps*

[17] S. Marovich. "GSC/Myrinet Device Adapter External Reference Specification." Hewlett-Packard Company, Palo Alto, CA, Oct. 1995. **HP Confidential**—this document cannot be distributed outside of Hewlett-Packard with appropriate non-disclosure agreements.

[18] R. Martin. "HPAM: An Active Message layer for a Network of HP Workstations." In *Hot Interconnects II*, Stanford University, Stanford, CA, Aug. 1994.

[19] R. Martin, L. Liu, V. Makhija, and D. Culler. "Lanai Active Message Release." Computer Science Division, University of California, Berkeley, Oct. 1996. *http://now.cs.berkeley.edu/AM/lam_release.html*

[20] Myricom, Inc. "LANai 4.X Specification." Jan. 1996.

[21] S. Pakin, M. Lauria, and A. Chien. "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet." In *Supercomputing '95*, San Diego, CA, 1995.

[22] "PA-RISC I/O Architecture Specification," ver. 0.95. Hewlett-Packard Company, Palo Alto, CA, July 1994. **HP Confidential**—this document cannot be distributed outside of Hewlett-Packard with appropriate non-disclosure agreements.

[23] S. Rodrigues, T. Anderson, and D. Culler. "High-Performance Local-Area Communication With Fast Sockets." In *Proceedings of the 1997 USENIX Conference*, Anaheim, CA, Jan. 1997.

[24] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. "Myrinet: a Gigabit-per-Second Local Area Network." *IEEE Micro*, Feb. 1995.

[25] C. Seitz, N. Boden, J. Seizovic, and W. Su. "The Design of the Caltech Mosaic C Processor." In *Proceedings of the University of Washington Symposium on Integrated Systems*, 1993.

[26] "Slipstream External Reference Specification," rev. 2.3. Hewlett-Packard Company, Palo Alto, CA, Jan. 1995. **HP Confidential**—this document cannot be distributed outside of Hewlett-Packard with appropriate non-disclosure agreements.

[27] J. Smith and M. Primmer. "Tachyon: a Gigabit Fibre Channel Protocol Chip." *Hewlett-Packard Journal*, Oct. 1996.

[28] Sun Microsystems SparcStation-20 Workstation.

[29] "U2 External Reference Specification," ver. 5.0. Hewlett-Packard Company, Palo Alto, CA, Jan. 1994. **HP Confidential**—this document cannot be distributed outside of Hewlett-Packard with appropriate non-disclosure agreements.

[30] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. "Active Messages: a Mechanism for Integrated Communication and Computation." In *Proceedings of the 19th International Symposium on Computer Architecture (ISCA '92)*, Gold Coast, Australia, May 1992.