SUMMARY

Cluster computing is an alternative approach to supercomputing where a large number of commodity workstations are utilized as the processing elements in a multiprocessor system. These workstations are interconnected by high-performance system area network hardware and specially designed "message layer" communication software. In the current generation of cluster computers, researchers have optimized message layers for communication between the host CPUs in the cluster in order to provide scalable computing performance. However, the recent development of a number of high-performance peripheral devices challenges the notion that message layers should be designed in such a CPU-centric manner. Modern peripheral devices feature powerful embedded processing and storage capabilities that can be leveraged to boost the performance of distributed applications. These peripherals function as sources and sinks of application data, and in some cases, as computational accelerators for offloading host-CPU tasks.

As Moore's Law continues its relentless trend, there will continue to be a migration of computing power to peripheral devices. Future clusters will not appear anything like the clusters of today. They will be rich in connectivity and computing power that is deeply embedded in the distributed components of the cluster. We refer to this new generation of systems as resource-rich cluster computers (Figure 1). These systems differ from traditional clusters in that application processing takes place in both the host CPUs and the peripheral devices. While the semiconductor industry continues to alter the economies of scale, the system software that productively enables resource-rich clusters is sorely lagging. Specifically, current generation message layers are ill equipped to service the needs of resource-rich

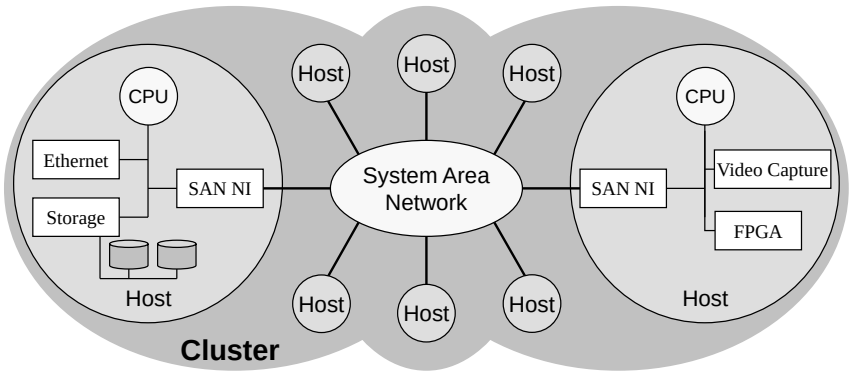

Figure 1: The architecture of emerging resource-rich cluster computers.

clusters, as they are not designed to utilize peripheral devices as globally accessible resources in a cluster. This thesis focuses on the challenge of designing extensible message layers for this new generation of resource-rich clusters. We are specifically concerned with making peripheral devices available as globally accessible resources in the context of a programming model that permits applications to effectively and efficiently exploit the capabilities afforded by resource-rich clusters. The key contributions of this thesis fall into two categories. The first includes design concepts and programming abstractions for structuring messages layers to integrate powerful peripheral devices into a globally accessible pool of resources. The second class of contributions is engineering solutions to the challenging problems of effectively and efficiently realizing these design concepts in a manner that tracks the evolution of technology, that is, the continued migration of computing power to distributed resources.

List of Acronyms

3GIO:	Third generation I/O
AGP:	Accelerated graphics port
AM:	Active messages
API:	Application programming interface
ASAN:	Active system area network
ASIC:	Application-specific integrated circuit
BIP:	Built-in Parallelism message layer
BT8x8:	Brooktree video capture card chipset
COTS:	Commercial off-the-shelf
CPLD:	Complex Programmable Logic Device
CPU:	Central processing unit
DMA:	Direct memory access
DSM:	Distributed shared memory
DSP:	Discrete signal processor/processing
EISA:	Enhanced industry standard architecture
FC:	Flow control
FFT:	Fast Fourier transform
FM:	Fast messages
FPGA:	Field-programmable gate array
GM:	Glenn's Messages
GNU:	GNU's not Unix
GRIM:	General-purpose Reliable In-order Messages
IB:	InfiniBand
IP:	Internet Protocol
ISA:	Industry standard architecture
LC:	Logical channel
LFC:	Link-level flow control
MCA:	Microchannel adaptor
MFLOPS:	Millions of floating-point operations per second
MP:	Multiprocessor
MPI:	Message Passing Interface
MPP:	Massively parallel processor
MTU:	Maximum transfer unit
NI:	Network interface
NIC:	Network interface card
PC:	Personal computer
PCI:	Peripheral component interconnect
RAID:	Redundant array of independent disks
RPC:	Remote procedure call
SAN:	System area network (also storage area network)
SMP:	Symmetric multiprocessor
SRAM:	Synchronous RAM
TCP:	Transmission control protocol
TPIL:	Tunable PCI injection library

UDP:	User datagram protocol
VM:	Virtual memory
VMMC:	Virtual memory mapped communication
VNN:	Virtual node number

CHAPTER I

INTRODUCTION

1.1 Background

After years of escalating supercomputer costs, a number of researchers in the early 1990's began investigating alternative means of constructing high-performance computing platforms that could satisfy the needs of both commercial and scientific parallel-processing applications. One of the most successful results of this effort is the field of cluster computing. In cluster computing a large number of commercial workstations are interconnected with a high-performance communication network so that the workstations can function as the processing elements of a large parallel-processing machine. While cluster computers typically lack the peak performance levels of traditional supercomputers, they provide an excellent cost-to-performance ratio that has attracted the attention of many users.

A key technology that makes cluster computers possible is the message layer software that implements inter-processor communication within the cluster. This software provides a set of message-passing programming abstractions that are utilized to transport data between communication endpoints in the cluster. Early message layer research efforts discovered that end application performance is often sensitive to the latency and bandwidth characteristics of a message layer's implementation. Therefore a significant amount of research during the 1990's focused on improving the host-to-host communication performance of a cluster's message layer. This effort has resulted in message layers that are highly optimized for transferring data between a cluster's host CPUs.

However, modern workstations are designed and optimized for high-speed sequential computation while accessing relatively slow peripheral devices. They are not optimized for inter-processor communication. Current message layers are designed to be as efficient as possible given these constraints and optimize transfers between host CPUs. The advent of powerful, inexpensive embedded processors has produced a migration of computing power to the peripheral devices "closer" to the sources and sinks of data. Media servers, content processing clusters, and data-intensive scientific applications all rely on complex interactions with peripheral devices to complete their objectives. These cards intelligently manage network and disk activities on behalf of the operating system to reduce the workload of the host. Other manufacturers have constructed powerful accelerator cards that utilize specialized hardware to accelerate the computational performance of certain operations. A cluster node now is comprised of multiple relatively powerful CPUs interspersed between peripherals, high-speed networks, and low speed intra-processor buses. CPU-centric message layers ignore this migration of compute power and no longer effectively use communication resources.

The inclusion of powerful peripheral devices into the cluster results in a new class of clusters which we refer to as *resource-rich cluster computers*. In these systems application processing takes places in both the host CPUs and peripheral devices. Traditional CPU-centric communication libraries are rapidly becoming a bottleneck and they do not provide the fundamental mechanisms that allow a peripheral device to be efficiently utilized as a resource in the cluster's distributed environment.

The goal of this thesis is to investigate, implement, evaluate, and deliver a set of communication abstractions and the associated message layer for resource-rich clusters governed by the principle of extensibility. In one dimension, extensibility refers to the ability to easily add new peripheral devices to the message layer as sources and sinks of data. A second dimension of extensibility refers to the ability to easily support multiple higher-level abstractions, e.g., sockets.

1.2 The Thesis

The work presented in this thesis addresses this problem by defining key aspects of a message layer that allow it to serve as a flexible means of interconnecting diverse endpoints in a cluster. A central part of this work is the migration of core message layer functionality into an intelligent network interface (NI) card. Performing management functions in the NI simplifies the amount of work an endpoint must perform to interact with other cluster resources, which in turn makes it easier to integrate new peripheral devices into the cluster. Migrating functionality into the NI is also beneficial because it allows the message layer to be utilized in an extensible manner. Extensible in this case refers to the ability for end users to layer new functionality on top of the core message layer and utilize the software in new and creative manners.

In this work we define three specific characteristics of a message layer required to support the inter-processor communication needs of resource-rich clusters.

- **Reliability:** The transfer of data from one endpoint to another is reliably managed using per-hop flow control. In this approach data moves from one stage to the next in the communication path as buffer space becomes available. This approach removes the need for end-to-end flow control being managed by endpoints that have historically resided in host CPUs.
- **Virtualization of Resources:** Endpoints and the communication between them should be decoupled from the underlying physical channels and hardware. Thus the message layer is designed to support multiple logical channels of traffic. These channels solve problems with multiple endpoints sharing a single NI and allow for different traffic streams to be insulated from each other.
- **Abstractions:** Finally, the message layer is equipped with both active message and remote memory programming interfaces. The active message interface allows remote CPUs and peripheral devices to be controlled with a flexible API while the remote memory interface allows large blocks of data to be transferred at high speeds between endpoints.

The General-purpose Reliable In-order Message layer (GRIM) has been constructed as a means of investigating the low-level communication characteristics of a resource-rich cluster. GRIM implements the functionality described in this thesis for a commodity x86-based cluster of hosts interconnected by a high-performance Myrinet network. To demonstrate extensibility, GRIM has been utilized for interactions with four different types of commercial peripheral devices in the cluster: an intelligent LAN and storage adaptor card, an FPGA accelerator card, a video capture card, and a generic video display device. Additionally,

to demonstrate the extensible nature of the core GRIM library, it has been extended with functionality to support multicast operations in the NI and provide users with a TCP sockets programming interface. While other message layers may exhibit similarities to some of GRIM's functionality, GRIM is the only (known) message layer that supports direct interactions with peripheral devices in a portable manner.

1.3 Organization of the Dissertation

The work presented in this thesis is organized as follows.

- **Chapter 2**: A brief background of cluster computers is provided to summarize how clusters have emerged and evolved over the last decade. Fundamental description of traditional cluster hardware is presented, as well as brief descriptions of existing communication libraries for cluster computers.
- **Chapter 3**: This chapter provides information about the environmental characteristics of resource-rich clusters. Based on these characteristics fundamental properties of a communication library for these clusters is discussed.
- **Chapter 4**: The guidelines for designing a resource-rich cluster communication library are then applied to implement a real system. This chapter discusses the core functionality of the GRIM communication library.
- **Chapter 5**: The performance characteristics of GRIM for traditional transactions between host CPUs is examined and compared with existing work.
- **Chapter 6**: This chapter provides a description of how commercial peripheral devices can be attached to the GRIM communication library. As an example of the horizontal extensibility of GRIM, four commercial peripheral devices with different operating characteristics are integrated into the GRIM library. Performance measurements are provided for each device.
- **Chapter 7**: Integrating distributed, specialized computing resources into a unified infrastructure for an application is the topic of this chapter. Specifically, this chapter provides insight as to how peripheral devices can be utilized to construct distributed, computational pipelines.
- **Chapter 8**: To demonstrate the vertical extensibility of GRIM, this chapter provides implementation details of a multicast system that performs message replication in the NI, general-purpose fragmentation and reassembly mechanisms, and an emulation of a sockets API.
- **Chapter 9**: The thesis concludes with some summary remarks and directions for future work.

CHAPTER II

BACKGROUND

By the end of the 1980's, the need for high-performance computing platforms in scientific and military applications had resulted in the emergence of a small number of supercomputer companies. These companies constructed large-scale systems that utilized massive amounts of custom hardware to improve application performance. Unfortunately, because these systems were extremely expensive, supercomputers were not a practical option for a large number of end users. Therefore, researchers in the 1990's began exploring alternative high-performance computational platforms that could be constructed in a more cost-effective manner. One of the results of this effort is the field of *cluster computing*. In cluster computing a large number of commercial workstations are collectively utilized to function as a single, multiprocessor system. Since system hardware is comprised of widely available commercial components, cluster computers can be constructed at a fraction of the cost of traditional supercomputers. As such, a considerable amount of high-performance computing research in recent years has focused on improving cluster computer performance.

A key challenge in improving cluster computer performance is adapting commodity hardware and software to function as part of a high-performance, multiprocessor system. Early cluster computing efforts revealed that application performance is highly dependent on the performance of a cluster's communication facilities. From a hardware perspective, several companies have addressed this issue by constructing system area networks (SANs) that provide an order of magnitude improvement over traditional local area networks (LANs). From a software perspective, researchers have constructed specialized communication libraries, or message layers, that are designed to deliver native SAN performance to end applications. In addition to facilitating low-latency, high-bandwidth communication, these message layers provide a programming abstraction where the cluster is viewed as a pool of host CPUs in a large *virtual parallel-processing machine*. This abstraction has sufficed for numerous researchers to effectively utilize a cluster computer's hardware as a distributed multiprocessor system.

2.1 Evolution of High-Performance Computing Platforms

Supercomputers are the computational systems that deliver the highest peak performance of all computer systems available at a given point in time. These systems typically employ large amounts of custom hardware to accelerate computational performance and often feature specialized computer architectures. Supercomputers have been primarily designed to process complex scientific applications that frequently exhibit large amounts of data parallelism. A number of commercial supercomputer systems have been produced since early groundbreaking work performed by the industry in the 1970's. The evolution of this technology provides both insight into high-performance computing and a motivation to continue the work in related research areas.

2.1.1 A Brief History of Commercial Supercomputers

While numerous people have contributed to the field of supercomputing over the years, perhaps the most influential individual in this effort is pioneer Seymour Cray. After leaving the Control Data Corporation in 1972 to form Cray Research, Cray began work on a new computer architecture that would provide significant gains in peak performance levels. In addition to advances in high-speed circuitry, Cray investigated the use of sophisticated vector processing units that allow computations to be applied to a stream of data to achieve high throughput. In 1976 the Cray-1 [81] was brought to market with a retail value of approximately nine million dollars and a record-breaking performance of 133 million floating-point operations per second (megaflops). In addition to being a technological marvel, the Cray-1 demonstrated that there was a definite market for expensive high-performance computing systems. Cray continued his work with vector processor systems, producing the 2 gigaflops Cray-2 in 1985 and the 5 gigaflops Cray-3 in 1989. A number of other computers followed the trend of vector processor systems, including the Meiko CS-2 [59], the NEC SX series supercomputer [60], the Fujitsu VP series supercomputer [91], and IBM's vector extensions to the System/370 [69]. Currently, the fastest system in the world [32] is the NEC SX-6, used in the Earth Simulator Center [102] in Japan. This system provides up to 8 teraflops of performance and employs multiple single-chip vector processing units.

The supercomputing industry also explored other architectural techniques for increasing the computational performance of a system. A key effort in this work is the use of a large number of processors to perform computations in parallel. In the SIMD (single instruction stream, multiple data streams) approach, a large number of identical processors perform the same series of operations on different data sets. Multiple SIMD systems were constructed in the early 1990's, including the MasPar Computer Corporation's MP-1 [17] and the Thinking Machines Corporation's CM-2 [41]. Both of these systems housed up to 16,384 SIMD processing elements, and could be used for parallel applications such as image processing. Due to the programming complexity of SIMD, researchers began constructing MIMD (multiple instruction streams, multiple data streams) systems that employed a large number of general-purpose CPUs. This work resulted in massively parallel processing (MPP) systems such as the Intel Paragon [45] (up to 4,000 Intel 80860 processors), the TMC CM-5 [55] (up to 16,000 SUN SPARC processors), and the Cray Research Cray-T3E [83] (up to 2,048 DEC Alpha 21164 processors).

2.1.2 Motivation for Alternate Computing Platforms

While the supercomputer companies of the 1980's provided significant advances in the field of high-performance computing, a large number of these companies withdrew from the supercomputer business in the 1990's. In hindsight it can be said that a common vulnerability for these companies was the large amount of custom design that was required to build a supercomputer. Several of these companies operated with a vertical design methodology, constructing all components of the system from the individual processors to the interconnection network. While having complete control over the design space gave engineers freedom to innovate performance enhancements, product design times were increased and complicated by the volume of custom hardware design that was required. Therefore, new supercomputers were expensive, brought to market infrequently, and often could not be designed in time to utilize the latest developments in state-of-the-art technology.

Additional issues make traditional supercomputers less appealing to researchers that

need high-performance computing platforms. First, supercomputers generally are not scalable and therefore offer a limited lifetime of leading-edge use. An investment in a state-of-the-art supercomputer depreciates rapidly in value due to Moore's Law, thereby making current systems obsolete within 18 months. Second, supercomputers require specialized hardware and software maintenance that adds to the expense of ownership. These components can be expensive to replace and there are generally few people that are trained to perform such maintenance. Finally, it must be noted that a risk in purchasing a traditional supercomputer is that the manufacturer might go out of business or otherwise abandon support for a particular product. Maintaining and utilizing orphaned hardware is time consuming and ultimately impedes end users.

Given the problems associated with using traditional supercomputers, a number of researchers in the early 1990's began exploring alternative methods by which high-performance computational platforms could be constructed. This effort made several observations about commercial technological advances and the global marketplace that would influence the construction of future parallel-processing systems. These observations include the following:

- **Commercial Off-the-Shelf (COTS) Parts**: In industry there are numerous corporations producing state-of-the-art hardware and software components. By using COTS parts, designers leverage other people's work and reduce the design time for a system. COTS parts are also beneficial because components can easily be replaced or upgraded from third-party products.

- **Growth in the Workstation and Network Markets**: Consumer demand for personal computers has resulted in high-performance workstations that are available at a low cost. Processor design in this market remains competitive, resulting in frequent updates to peak performance levels. Likewise, consumer interest in the Internet has resulted in advances in network hardware. The need for faster networks has resulted in low-cost local area networks (LANs) that economically offer high-bandwidth communication.

- **A Rich Software Environment**: An important aspect of commodity workstations is the wide availability of software. Operating systems such as GNU/Linux provide a UNIX-like environment with built-in network features. The open source nature of Linux allows researchers to easily incorporate custom functionality into the operating system kernel.

In summary, researchers observed that advances made in consumer markets in the 1980's and 1990's had resulted in hardware that was widely available, economical, and offered respectable levels of computational performance. These systems could be utilized to provide impressive price-to-performance ratios and have benefited from considerable efforts to improve the PC's software environment.

2.1.3 Emergence of Cluster Computers

In the mid-1990's, researchers began investigating the use of multiple commodity workstations to construct a new form of high-performance system. This work resulted in the notion of a *cluster computer*, where a number of workstations are collectively utilized to function as a single parallel-processing system. Through commodity network hardware and specialized communication software, a cluster computer can effectively appear as a large pool of

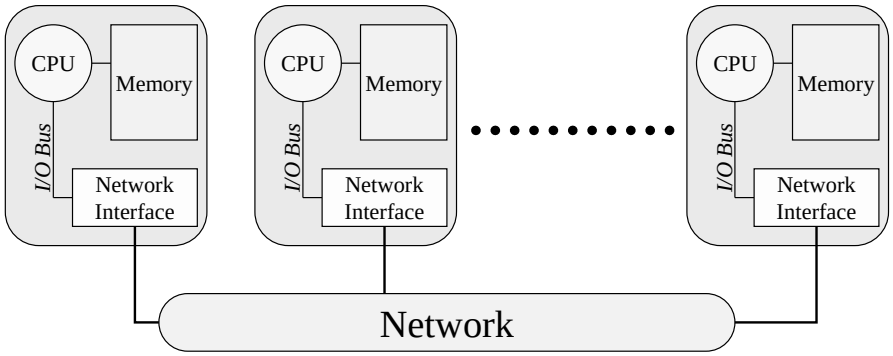

Figure 2.1: A cluster computer constructed with commodity workstations and network hardware.

host processors to the end user. Since workstations in the cluster are commercially available products, cluster computing can leverage the performance gains achieved by the workstation industry. The high-level architecture for a cluster computer is depicted in Figure 2.1.

One of the first cluster computing projects to receive serious attention from the scientific community was NASA's Beowulf Project [14]. In this work, researchers demonstrated that a small number of dedicated workstations could collectively operate to perform computations that were beneficial to scientific computing [13]. Utilizing commodity PCs equipped with multiple Ethernet adaptors, the 16-node demonstration cluster achieved 60 megaflops in 1994. Later clusters in this project would expand the number of workstations to 199 nodes and accomplish 10 gigaflops of performance for under $50,000. While researchers stated that Beowulf clusters were a far step from true supercomputing, the price-to-performance ratio was a significant motivator for building such clusters. After this work numerous research institutes constructed Beowulf-style clusters out of Ethernet-connected PCs.

An observation made about the early Beowulf-style of cluster was that while some applications performed well, others did not. An examination of this problem revealed that these clusters were severely limited in terms of communication performance. Grossly parallel applications that did not require significant amounts of communication between host CPUs performed well because each task in the cluster could operate independently. However, applications that required frequent exchanges of data between CPUs performed poorly due to the low performance of the network. The conclusion to be drawn from this observation is that ultimately, the communication performance of the cluster determines the granularity at which parallel-processing applications can productively use a cluster.

Realizing that the poor communication performance limited the types of applications a Beowulf cluster could run, researchers in the mid to late 1990's began examining ways in which the cluster's communication performance could be enhanced. Several academic projects focused on adding hardware to facilitate specific types of communication. In the SHRIMP project at Princeton [18], workstations were extended with hardware that allowed hosts to operate in a distributed shared memory (DSM) environment. At Purdue, the PAPERS project [31] utilized custom hardware to rapidly distribute barrier synchronization information to host computers. However, the most significant advance for cluster computers came with the advent of commercially available system area networks (SANs). SANs provide communication performance that is over an order of magnitude better than traditional LANs. This allows for significant improvements in fine-grain parallel processing

performance. Current work in high-performance cluster computers involves delivering as much native performance from a SAN as possible to end applications.

2.2 Using Workstations as a Cluster Computer's Processing Elements

Multiprocessor systems are generally comprised of two types of hardware components: processing elements that are used to perform computations, and a communication network to distribute data in the system. In cluster computers, individual workstations function as processing elements, while commodity SAN hardware performs communication tasks.

2.2.1 Workstation CPUs

A number of vendors have constructed different workstations that can be utilized in a cluster computer. Historically, companies such as Sun Microsystems, Hewlett-Packard, SGI, and Compaq/DEC have dominated the workstation industry with CPU architectures that offer high performance at a relatively high cost. However, the workstation market for these companies has eroded over the last decade as x86-based PCs and PowerPC-based Apple Macintosh computers have steadily improved in performance and popularity. Because of its impressive price-to-performance ratio, the x86 PC has become the workstation of choice for the majority of cluster computing efforts. Therefore, the work presented in this thesis specifically deals with clusters constructed from x86-based PCs.

While affordable, x86-based systems have some of the most limiting architectural characteristics of any workstation when used for high-performance computing. First, the x86 is based on a 32-bit architecture that may not be sufficient for the processing needs of scientific applications that require 64-bit computations. Second, an x86 processor can only support 4 GB of physical memory. This trait limits the amount of state information an application can have loaded at a workstation, and is becoming more of an issue as memory prices continue to decline. Finally, in order to obtain peak performance levels in x86-based hosts, it is often necessary to utilize architectural extensions such as the MMX and streaming SIMD (SSE) units. The performance of these units can vary greatly between different generations of x86 processor.

2.2.2 Evolution of Workstation I/O Systems

A second key factor that affects the performance of a workstation as a processing element is the architecture of its I/O system. In ideal multiprocessor systems, processing elements are placed in close proximity to the NIs in order to allow fine-grained interactions between applications and the network. Unfortunately, in most workstations the CPU and NI are separated by a complex general-purpose I/O system. Transactions involving the I/O system can be up to an order of magnitude slower than similar transactions with host memory. While the computer industry makes improvements to x86 CPU performance multiple times a year, PC I/O performance is improved on average once every *three years*.

Figure 2.2 highlights the history of I/O systems utilized in PCs. The peripheral component interconnect (PCI) standard [71] provides reasonable performance and has become the de facto standard for peripheral devices in current workstations. While this thesis targets PCI-based systems, the implementation can be tuned to platforms with higher bandwidth I/O systems.

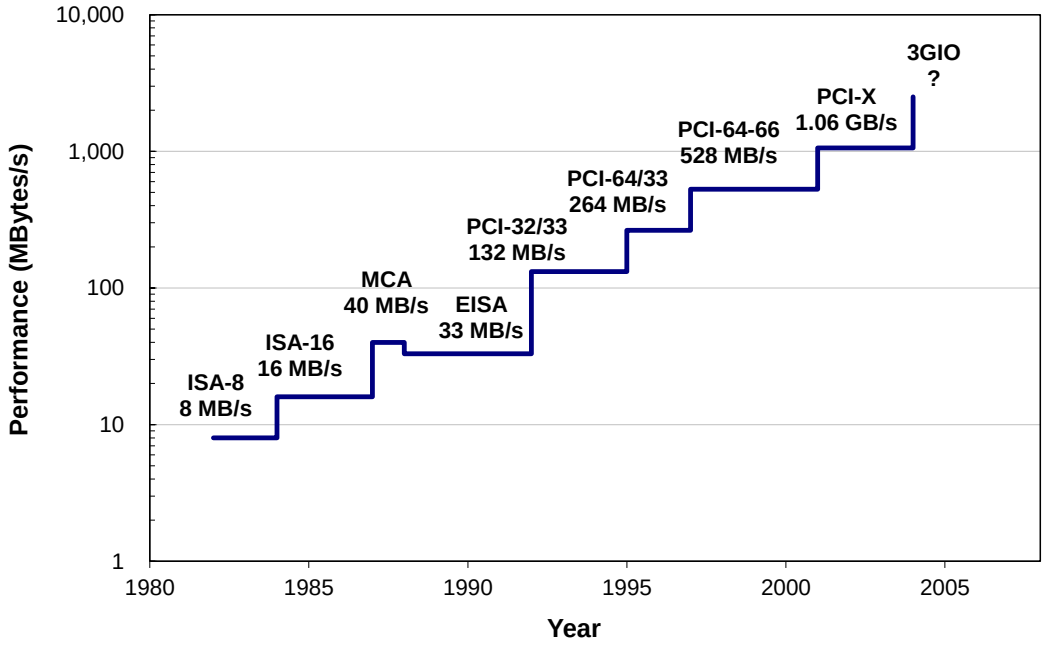

Figure 2.2: A history of PC I/O systems.

2.2.3 Peripheral Component Interconnect (PCI)

The peripheral component interconnect (PCI) standard was introduced in 1992 as a means of allowing high-speed peripheral devices to be incorporated into the x86 PC architecture. The architecture of modern host systems employing PCI is depicted in Figure 2.3. In this architecture the system's memory controller is responsible for routing data between the host CPU(s), main memory, and peripheral devices on the PCI bus. At boot time the memory controller assigns regions of the host's 32-bit physical address space to both main memory and individual peripheral devices. When a device driver for a PCI card is loaded into the kernel, the driver can establish a memory translation that allows the card's memory to appear in the kernel's virtual address space. The driver can then share this mapping with user-space applications through the implementation of a memory map system call, handled by the device driver. Doing so allows user-space applications to directly read and write the on-card memory of a peripheral device.

In addition to memory mapped reads and writes from the host CPU, communication involving peripheral devices can be facilitated by on-card DMA engines that are available with *bus-mastering* PCI devices. These DMA engines adhere to the low-level PCI bus standard and can be used to transfer blocks of data between a peripheral device and host memory or other peripheral devices in the system. All memory references on the PCI bus are in terms of the host's 32-bit physical address space in the x86 architecture. Each PCI device also controls an interrupt request (IRQ) line, which can be used to transmit an interrupt to the host CPU. Due to a limited number of IRQs in a host, multiple peripheral devices may share the same interrupt, requiring each card's device driver to determine which card initiated an interrupt.

A number of modern PCI devices support sophisticated DMA transfers through the use of chained DMA operations. With chained DMA, a peripheral device is capable of performing a series of DMA operations as specified by a linked-list of DMA descriptors.

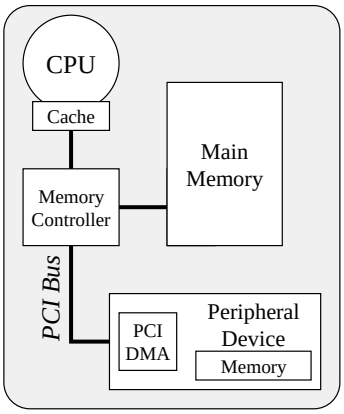

Figure 2.3: Architecture of a modern host utilizing PCI.

Each descriptor in a linked list specifies the length, direction, and addresses of a transfer. Some implementations allow users to specify whether an interrupt should be generated for the host at the completion of a transfer for a given DMA descriptor. The DMA engine processes each descriptor linearly until an end-of-chain marker is specified in a descriptor. While most cards employ a similar API for controlling chained DMA operations, it should be noted that there is no standard and that each peripheral device driver must be outfitted with custom functionality.

2.2.4 Architecture Tradeoffs

There are a number of architectural tradeoffs designers must face when considering how cluster computer workstations can be used as processing elements in a multiprocessor system. From the previous three subsections it is clear that host CPUs in a cluster computer incur significant overheads when interacting with other CPUs in the cluster. This trait is a serious obstacle for application designers, especially when clusters are compared to traditional MPP supercomputers that allow fine-grained network interactions. However, a workstation by itself is a complete, self-contained system that features processing, memory, and storage resources, as well as a sophisticated operating system for managing these resources. Therefore, a processing element in a cluster computer is more likely to be better equipped to perform diverse tasks than a processing element in a traditional MPP supercomputer. The architectural tradeoffs of using workstations as processing elements therefore suggests that cluster computers are better utilized for computations where operations can localized to individual processing elements.

2.3 Cluster Computer Network Hardware

In addition to processing elements, multiprocessor systems must be equipped with communication infrastructure that allows distributed processor elements to interact. In cluster computers, this infrastructure is built from commercial network hardware. Several network substrates have been used in cluster computers over the years. One of the most popular approaches is to employ traditional LAN hardware such as Ethernet. While economical, the drawback of Ethernet is that it only offers limited host-to-host communication performance in a cluster environment. Therefore, a number of companies have constructed system area

network (SAN) products that are better suited for cluster computers. These SANs feature multi-gigabit bandwidths and host-to-host transmission latencies that are less than 50 μs. SANs generally offer high levels of reliability and commonly utilize intelligent NI cards to manage network interactions. Examples of SANs include Myricom's Myrinet [19], Compaq's Servernet [42], Dolphin Interconnect Solutions implementation of the scalable coherent interconnect (SCI) [2], and Quadrics' QsNet [72].

2.3.1 Ethernet

The Ethernet network standard first created at Xerox Parc labs in 1976 [62] has grown to become the most popular network ever utilized. The Ethernet standard has been periodically updated over the years, and now features link speeds of up to 10 Gbps in the most recent standard [8]. Ethernet NI cards traditionally have employed a simple hardware architecture where the NI only manages a pair of message queues for incoming and outgoing transmissions. In this approach the host CPU formats and processes all messages transferred to and from the network. In order to reduce the workload of the host CPU, some high-end Ethernet NI cards feature more sophisticated processing engines that are capable of managing network interactions on behalf of the host. These intelligent NI cards are especially beneficial for Gigabit Ethernet networks where high-bandwidth transactions are necessary.

While widely available, Ethernet is not the ideal communication substrate for cluster computers. The primary issue is that Ethernet was designed for use in LANs. Since data in LANs is transmitted over long distances, Ethernet is largely optimized for bandwidth but not latency. Another consequence of Ethernet being designed for LANs is that the hardware is not designed to facilitate reliable transmissions. Instead, workstations in the cluster must implement reliable transmission protocols that can tolerate dropped messages. Finally, Ethernet hardware can be criticized because currently there is a lack of high-performance NI adaptors. In [40] researchers compare several commercial Gigabit Ethernet adaptors. Tests using a host with 32b/33MHz PCI found that the maximum obtained bandwidth was only 436 Mb/s, while most of the cards provided less than 200 Mb/s. In tests using a host with 64b/66MHz PCI general performance rose to 650 Mb/s, with one card obtaining 928 Mb/s. While industry is steadily improving Gigabit Ethernet product performance, these tests demonstrate that it is still challenging to obtain the peak performance levels of the network.

2.3.2 Scalable Coherent Interconnect (SCI)

The scalable coherent interconnect (SCI) standard is a SAN for clusters that has gained widespread use in Europe. SCI evolved out of the Futurebus+ project [1] in 1988 as a means of developing a next-generation I/O infrastructure for high-performance workstations. SCI is designed to allow a large number of hosts to function as part of a distributed shared memory machine. In the programming model for this system, each host is allocated a region of memory in SCI's global address space. When a host reads or writes a region of the address space that is not available at the local node, the SCI NI card forwards the transaction to the memory system of the host that owns the memory. Distributed memory interactions take place efficiently in SCI because shared memory protocols are implemented in hardware in the SCI NI cards. Initial versions of SCI interconnected hosts in ring topologies similar to token ring LANs. As the standard evolved SCI hardware was adapted to operate in

Table 2.1: A history of Myrinet network interface cards.

Year	Processor	Clock Rate	Memory	Host I/O	Link Speed
1994	LANai 3	25 MHz	128 KB	20 MHz SBUS	640 Mb/s
1996	LANai 4	33 MHz	1 MB	32b/33 MHz PCI	1.28 Gb/s
2000	LANai 9	100-200 MHz	2-8 MB	64b/66 MHz PCI	1.28 Gb/s 2.0 Gb/s

point-to-point network topologies using dedicated switches.

An advantage of SCI's approach to communication is that it provides a specific set of actions that the network hardware must perform. These actions can be implemented with custom hardware that benefits from circuit-level optimizations. While this prevents the NI from being extended with functionality by the user, it allows NI hardware to be simplified and produced more economically. One of the largest vendors of SCI hardware is Dolphin Interconnect [6]. This company's implementation of SCI has an application-to-application performance of up to 2.6 Gb/s in bandwidth and 1.4 μs in latency [7] (using IA64 Itanium hosts).

2.3.3 Myrinet

Myricom's Myrinet is one of the most commonly utilized SANs for cluster computers due to its high levels of performance and programmability. Myrinet is a descendent of the Mosaic [84] and ATOMIC [35] research projects. In these projects researchers developed a high-performance network for multiprocessor systems that employed source-routed, wormhole [29] messages to reduce switch latencies. These networks provided high levels of data reliability and would only drop messages if deadlock was suspected. Myricom converted Mosaic into a commercial product known as Myrinet for use with commodity workstations. Myrinet in its current implementation consists of network switches, 1.28+1.28 to 2.0+2.0 Gb/s links, and programmable NI cards. Network hardware is connected in a point-to-point fashion, allowing the construction of both regular and irregular network topologies. In minimizing switch latency, Myrinet designers have pushed network tasks out of switch hardware and into the NI cards. A beneficial side effect of this design choice is that network functionality (e.g., multicast or added fault tolerance) can be implemented by users in the form of NI firmware. Myricom has released several generations of NI hardware as summarized in Table 2.1.

The organization of the Myrinet NI is depicted in Figure 2.4. In this architecture the NI is situated between an interface to the host I/O system and an interface to the network wire. High-speed SRAM is utilized to house both the executable firmware and data for the NI. Firmware typically occupies less than 256KB of SRAM memory, allowing the remaining memory to be used as needed by communication library designers. The SRAM is shared between the LANai and DMA engines through a priority based memory controller.

While the architecture of the Myrinet NI is relatively simple, the NI is a powerful device because it can support multiple data transfers at the same time. The NI can be configured to simultaneously send data to the network, receive data from the network, and issue a DMA transfer to or from host memory. In more recent versions of the Myrinet NI the NI is also capable of supporting multiple DMA transfers between the NI and host using four

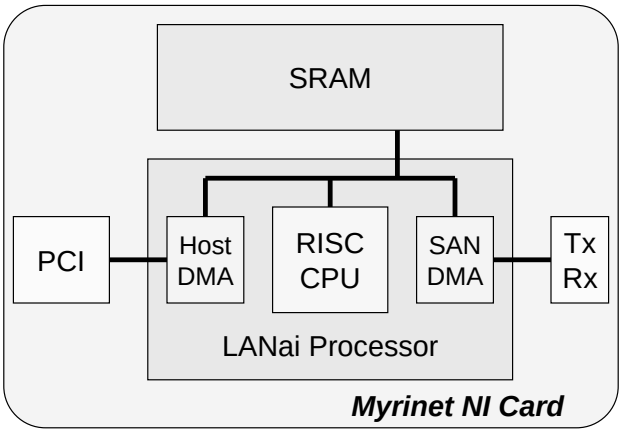

Figure 2.4: Architecture of the Myrinet NI card.

PCI DMA engines. The programmable nature of the NI has allowed firmware designers to construct efficient communication pipelines with the NI, where data is transferred in a cut-through manner without buffering delays in the NI. Basic performance measurements of the LANai 4 and 9 NI cards are provided in Appendix A.

2.3.4 Quadrics QsNet

The QsNet [72] interconnection network is a relatively new SAN product created by Quadrics in Europe. QsNet is currently being utilized in high-end cluster computers such as the Terascale Computing System [73] at the Pittsburgh supercomputer center (currently the third fastest supercomputer in the world [32]). Similar to Myrinet QsNet uses wormhole routing to efficiently transfer data between NI cards through a point-to-point network. However, QsNet differs in that communication resembles a virtual circuit approach as wormhole transmission paths are not released until the receiver transmits an acknowledgement token. Similar to SCI QsNet provides a means of allowing hosts in the network to share a global address space. In order to accelerate remote memory operations, NI cards are equipped with hardware engines that can dynamically translate virtual addresses into physical addresses. Initial reports of QsNet indicate that it is capable of providing over 2.4 Gb/s of bandwidth and approximately 2 μs of latency between user-space applications.

2.4 Cluster Computer Network Software

After early work in Beowulf-style clusters, researchers observed that the high latency of traditional communication libraries for LANs precluded fine-grained cluster applications. Given the raw performance available in SAN hardware, a considerable amount of cluster computing research in the late 1990's focused on techniques for harnessing this communication performance. This work resulted in the development of a number of custom communication libraries or message layers that offered mechanisms for reducing communication latency and increasing bandwidth between host-level applications. The most commonly utilized SAN in this effort is Myrinet due to its open source software and well-documented hardware. A large number of message layer packages have been implemented for Myrinet, including Active Messages (AM, AM II) [23],[23], Fast Messages (FM) [70], PM [89], Link-level Flow Control (LFC) [12], Trapeze [101], Virtual Memory Mapped Communication

(VMMC) [33], GM [64], and BIP [75].

2.4.1 Limitations of LAN Protocols

Early Beowulf-style cluster computers utilized traditional LAN hardware and software to provide reliable communication between workstations in the cluster. These clusters typically employed Ethernet network hardware and communication software based on the transmission control protocol (TCP). While leveraging existing LAN equipment allowed large cluster to be constructed easily in a cost-effective manner, researchers observed that these Beowulf-style clusters offered limited performance in some parallel processing applications. The fundamental issue observed with using LAN equipment is that it is primarily designed to transmit data over long distances using an error-prone medium. Therefore LAN software such as TCP must perform a number of complex transmission management operations in order to guarantee that messages are reliably delivered in the proper order to a destination.

Cluster computers have different operating conditions than LANs. In cluster computers, workstations are separated by small distances and utilize dedicated network switches for local communication. Under these conditions messages are dropped or reordered by the network infrequently. Therefore TCP's reliable transfer mechanisms are not optimal for cluster computers and are in general too heavy weight for high-performance applications. Transmissions using Ethernet and TCP can suffer communication latencies greater than 100 μs for host-to-host deliveries. By comparison, inter-processor communication in a symmetric multiprocessor (SMP) node takes place in only a few microseconds. This difference in communication performance is enough to significantly limit the effectiveness of cluster computers in the case of fine-grained, communication intensive parallel programs [58]. Therefore researchers in the late 1990's began investigating custom communication libraries or message layers that were better suited for cluster computers.

2.4.2 Message Layer Characteristics

Message layers for cluster computers serve as a means of transferring application data between *communication endpoints* in the cluster. Naturally there are many ways in which message layers can be designed. Therefore a first step in understanding how message layers function is to consider a few of the key characteristics of message layers. These characteristics include the following.

- **Programming Interface**: One of the most defining characteristics of a message layer is the programming interface that is provided to end users. Message layers generally employ one of three types of programming interface. First, active message [93] systems utilize an interface similar to remote procedure calls (RPCs) [16] where an application can invoke an operation at a remote endpoint simply by transmitting a message. Second, in rendezvous approaches sending and receiving endpoints are tightly synchronized and require the receiving endpoint to post requests to extract certain messages from the network. Finally, systems using a shared memory programming interface use remote memory operations to manipulate data located at different hosts in the cluster.

- **Buffer Management and Flow Control**: NI cards have a limited amount of buffer space for housing in-flight messages. Therefore an important aspect of a message layer is the means by which it manages the reliable transfer of messages from one endpoint

to another. In some message layers flow-control schemes are applied at either the host or NI levels in order to prevent messages from being dropped due to insufficient buffer space at the receiver. Other message layers do not implement such mechanisms, either for performance reasons or because they are not necessary. For example, in a shared memory system the receiving NI always accepts and processes an incoming message and therefore buffer management is not necessary.

- **Delivery Order**: In a strictly ordered system, messages are processed by a receiver in the same order they were injected into the network by the sender. When messages are dropped in the network, a message layer with ordered delivery performs retransmission and reordering to maintain consistent data flow. In systems where network messages can carry priorities, some message layers allow higher priority messages to bypass lower priority messages by relaxing ordered delivery constraints.

- **Receiver Notification**: Another characteristic of a communication library is the manner in which the receiving application is notified that a new message has arrived. In message layers that notify the receiving endpoint, either an interrupt mechanism or polling is utilized. While interrupts allow applications to interact with the message layer only when new data has arrived, the interrupts can take place at any time and are therefore challenging to manage. Polling techniques require application to periodically examine the message layer for new data, but can generally provide better performance than interrupts. Shared memory systems do not necessarily need to utilize any explicit form of notification as this task is implicitly performed by the receiver application.

2.4.3 Common Message Layer Optimizations

Researchers often utilize a number of common techniques for improving the performance of a message layer. One of the earliest and most widely used techniques is to construct a message layer in user-space. This technique is beneficial because it allows an application to interact with the NI card without invoking expensive system calls. Another common technique used in many message layers is to make use of the reliable nature of SAN hardware. Since SAN hardware can operate for months at a time without a single bit error, researchers often simplify message layer protocols by assuming the common case of error-free transmissions. Finally, with the observation that the host CPU is much more powerful than the NI processor, a number of researchers have minimized the amount of work NI processors perform in the message layer. While these optimizations have boosted performance in message layers, such approaches have resulted in *CPU-centric* message layers. As will be discussed in the following chapter, these message layers are inappropriate for resource-rich cluster computers which require both host CPUs and peripheral devices to function as communication endpoints.

2.4.4 Myrinet Message Layers

A number of message layers have been constructed for Myrinet over the years. An excellent survey of several of these message layers is provided in [15]. The more influential of these message layers are summarized as follows.

- **Active Messages (AM, AM-II) [28], [23]**: The active messages project was one of the first academic message layer packages for Myrinet hardware. In addition to

demonstrating the active message programming paradigm, AM software illustrated that communication libraries implemented in user space could provide significant performance improvements. AM utilizes host-based flow-control mechanisms to manage buffer space in the library.

- **Fast Messages (FM)** [70]: The FM library was released shortly after AM and extends AM concepts by providing mechanisms for increased performance, stability, and usability. FM utilizes an active message programming interface and includes mechanisms for registering and managing application function handlers. Another feature of FM is its ability to efficiently fragment and pipeline large message transmissions, which allowed for significant gains in communication performance. While FM originally employed NI-based flow-control mechanisms, these mechanisms were later deferred to the host due to poor NI performance.

- **BIP** [75]: The BIP message layer is an effort to construct a lean message layer that can provide high performance for higher-level programming interfaces such as MPI [61]. BIP uses a rendezvous communication model where a receiver must provide the NI with information that specifies where the NI should store a particular incoming message. BIP provides no reliability guarantees and has been reported to have the best communication performance of any Myrinet message layer.

- **Virtual Memory Mapped Communication (VMMC)** [18]: The VMMC layer is designed to support shared memory operations on cluster computers. In this software the library provides efficient means of transferring blocks of data from the virtual memory of one application to the virtual memory of another application located on a different host. These operations take place with remote DMA operations and require no flow-control mechanisms. VMMC NI firmware is equipped with mechanisms to perform virtual to physical address translation, as well as facilities to cache translation results.

- **GM** [64]: GM is an industrial strength message layer from Myricom that provides good performance and is supported on a wide variety of cluster platforms. Like BIP, GM utilizes a rendezvous programming interface that works well with MPI. GM provides rich functionality at both receiving and sending endpoints and uses callback functions to notify applications that message layer operations have completed. GM requires that all data transferred with the message library be loaded in a block memory that is registered with the library. Registered memory allows the NI to efficiently DMA data between the host and card, with virtual memory translation performed through a simple table lookup.

2.5 The Virtual Parallel-Processing Machine

In addition to providing low-level mechanisms for transferring data between cluster workstations, SAN message layers provide a programming abstraction that allows end users to control a cluster's computational resources. This abstraction presents the cluster as a *virtual parallel-processing machine* that is capable of running parallel and distributed applications. An example of such a virtual machine is presented in Figure 2.5. In this example the message layer maintains information about the workstations in a cluster and provides an interface where applications can globally reference any host CPU in the cluster. Therefore

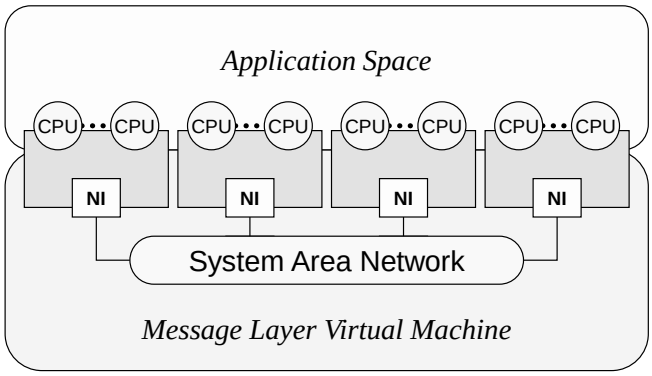

Figure 2.5: The virtual parallel-processing machine architecture provided to end users in current message layers.

an application running at one host CPU transmits data to another CPU by providing the message layer software with a message that is labeled with the reference identifier for the destination. In current generation message layers, host CPUs are the only resource included in the virtual machine architecture.

While there are many ways in which a virtual machine can be realized for a cluster, a common approach is to load each host in the cluster with an executable program that is part of an overall parallel-processing application. Each executable contains user-defined functionality for the local host as well as message layer library functions and information about the cluster's global resources. After all hosts in the cluster have executed message layer initialization functions, the virtual parallel-processing machine becomes operational and each host begins processing the application code defined in its local executable program. Maintaining the appearance of a virtual machine is a relative straightforward process in the message layer after this point, as the message layer must simply service application queries regarding cluster resources and route transmissions to the appropriate cluster resources.