

A Study of High Performance Communication
Using a Commodity Network of Parallel Computers

2000

Shinji Sumimoto

THE SUMMARY OF PH.D DISSERTATION

The increasing demands of information processing requires parallel computers using a number of computers connected with a high performance communication network.

Ten years ago, such a parallel computer could not be built without dedicated hardware. However, dramatic breakthroughs in the areas of microprocessor and computer network technology for personal computers (PCs) have made a parallel computer consisting of commodity PCs with high speed network possible. This parallel computer is able to achieve performance comparable to that of commercially available dedicated parallel computers. This kind of parallel computer is often called a “cluster system.”

However, existing communication facilities on a commodity network with a cluster system use mainly the TCP/IP protocol. The TCP/IP protocol can not ensure the maximum communication performance of network hardware.

The first purpose of this dissertation is to propose implementation designs which ensure the maximum communication performance on cluster systems using Gigabit Ethernet, one of the next generation of commodity networks. The second purpose of this dissertation is to show that communication using Gigabit Ethernet can achieve performance of application programs (application performance) comparable to that of dedicated cluster network communication.

To realize the first purpose, there are two types of approaches: approaches using hardware dependent designs, and approaches using hardware independent designs using existing network interface cards (NIC). In this dissertation, communication facilities using both of these approaches are designed, implemented, and evaluated.

The following hardware dependent designs are proposed: a reliable light-

weight communication protocol called "GoBack-N for PM" and design techniques which minimize information exchange cost based on a cost analysis of TCP/IP. There are significant design choices: where should a communication protocol be processed, in the host CPU or the NIC CPU; and where should the data structures for managing the send and receive buffers and triggering be allocated, on the host or the NIC memory. Since those structures are shared resources, the cost between host and NIC information exchange is crucial in performance.

The GigaE PM communication facility has been implemented using these designs on the Essential Gigabit Ethernet NIC. The performance benchmark results show that the communication bandwidth of GigaE PM is 1.7 times faster than that of the existing TCP/IP protocol, and the round trip time of GigaE PM is about one third that of TCP/IP. Therefore, the proposed method is effective in realizing high performance communication on Gigabit Ethernet.

In hardware independent designs, a design method to reduce reliable protocol processing overhead is proposed. The method is based on the results of a cost analysis of existing TCP/IP processing overhead. According to the analysis, the interrupt overhead caused performance degradation in addition to that of the TCP/IP protocol processing overhead. In order to eliminate the interrupt overhead, a communication protocol processing method has been proposed. This method does not use a software interrupt as existing protocols do. And the *Interrupt Reaping* technique has also been proposed. This technique eliminates the use of a hardware interrupt without modification of existing device drivers.

The PM/Ethernet communication facility has been implemented using these techniques on the Packet Engines G-NIC II Gigabit Ethernet NIC. The performance benchmark results show that the communication bandwidth of PM/Ethernet is 1.6 times faster than that of the existing TCP/IP protocol, and the round trip time of PM/Ethernet is about one third that of TCP/IP. These results show that proposed method is effective in eliminating two thirds of the TCP/IP processing cost.

As another hardware independent design, the *Network Trunking* technique

has been developed. It improves communication bandwidth using multiple NICs. The communication performance benchmark results show that the bandwidth performance of PM/Ethernet with four digital 100 BaseT NICs is 3.6 times faster than that with one 100 BaseT NIC.

To show that communication using Gigabit Ethernet achieves application performance comparable to that of dedicated cluster network communication, a communication facility which supports both a dedicated cluster network and a commodity network is used. As a result of the evaluation using the NAS parallel benchmarks, the performance of the IS benchmark on PM/Ethernet using Gigabit Ethernet achieves performance comparable to that on PM/Myrinet on a 16 node cluster. This result shows that practical high-performance cluster systems can be built using a commodity network.

Using the proposed method described above, it has been shown that a cluster system using a commodity network can achieve performance comparable to that of a cluster systems using a dedicated cluster network. It is expected that cluster systems using a commodity network will become more popular very soon.

Acknowledgement

I was given very valuable and insightful advice from Professor Norihisa Doi, Professor Kenichi Harada, and Professor Takashi Nodera. I would also like to thank Professor Hideharu Amano for his invaluable advice and warm support.

This thesis was completed while I was working at Real World Computing Partnership (RWCP). The head of RWCP, Dr. Junichi Shimada, gave me enough opportunities for research to last many years. Dr. Yutaka Ishikawa, the leader of the Parallel Distributed System Software Laboratory of RWCP kindly guided the author, who was very inexperienced, in the first tentative steps towards new technology. My interest was kindled by his passion for study.

It would not be possible to develop a high performance communication facility for a commodity network without the SCORE Cluster System Software which was developed by Dr. Atushi Hori, Mr. Hiroshi Tezuka, Mr. Hiroshi Harada and Mr. Toshiyuki Takahashi, researchers at the Parallel Distributed System Software Laboratory of RWCP.

I would like to thank Dr. Mitsuhiro Sato, the leader of the Parallel Distributed System Performance group at RWCP, and Dr. Tomohiro Kudoh, the leader of the Parallel Distributed System Architecture group at RWCP for their warm advice and vast store of knowledge of parallel computation and system architecture.

Mr. Noriyuki Soda of SRA Co., Ltd. gave a lot of advice on the basis of abundant Unix programming experience. Thanks to Mr. Toyohisa Kameyama, and Mr. Hiroshi Futsuhara, I found a comfortable program development environment ready.

I would like to express my appreciation for support from many people in Fujitsu, Ltd. and Fujitsu Laboratories, Ltd., especially Research Institute Managing Director, Mr. Hiromu Hayashi, Managers, Mr. Akira Jinzaki and Dr. Yasunori Kimura.

Last, and most important, I thank my dear wife Satoko and my sons Yuji and Koji, for continuous heartfelt support.

Contents

1	Introduction	1
1.1	The Need for Parallel Processing and Cluster Systems	2
1.2	Issues in Cluster Systems Using Commodity Networks	3
1.3	The Purposes of This Thesis	4
1.4	Overview of This Thesis	5
1.5	Contributions	5
2	Background	7
2.1	History of Commodity Hardware	8
2.2	The History of Massively Parallel Computers	12
2.3	Standardization of the Parallel Application Programming Environment	13
2.4	A History of Cluster Computing	14
2.4.1	Cluster Computing Using Commodity Networks in the Early 1990s	15
2.4.2	Cluster Computing Using Dedicated Cluster Networks	16
2.4.3	Cluster Computing Using Commodity Networks in the Late 1990s	21
2.5	Issues of Cluster Computing on Gigabit Class Commodity Networks	21
2.6	Summary of This Chapter	22
3	The RWC Project	24
3.1	The RWC Project	25
3.2	S _{Core} Cluster System Software	25
3.2.1	A Low Level Communication Facility: PM	27
3.2.2	A Multi-Threaded Language: MPC++	29
3.2.3	The MPICH-PM Communication Library	29

3.2.4	SCore-D	30
3.2.5	SCASH	30
3.3	The Cluster System Hardware Platform	30
3.3.1	RWC PC Cluster II	31
3.3.2	RWC SCore Cluster I	32
3.3.3	RWC SCore Cluster II	33
3.3.4	Application Performance on RWC Clusters	34
3.4	The SCore Software on Commodity Networks	35
3.5	Summary of This Chapter	36
4	Hardware Dependent Designs	37
4.1	Network Protocols and NICs	38
4.1.1	Cost Estimation	38
4.1.2	Discussion	40
4.2	Design Objectives of GigaE PM	40
4.3	GigaE PM Design	41
4.3.1	Virtual Networks and API	41
4.3.2	Where Should a Communication Protocol be Processed?	42
4.3.3	Reliable Communication	42
4.3.4	Information Exchange between the Host and the NIC	43
4.3.5	Host CPU Polling effects on PCI DMA transfer	45
4.4	GigaE PM Implementation	47
4.4.1	Overview of the GigaE PM Implementation	47
4.4.2	Overview of GigaE PM Message Handling	48
4.4.3	TCP/IP and Other Protocol Support	49
4.4.4	Comparison of Cost and Buffer Usage	49
4.5	GigaE PM Evaluation	50
4.5.1	Evaluation Environment	51
4.5.2	PM Bandwidth	51
4.5.3	PM Round Trip Latency	51

4.5.4	NAS Parallel Benchmarks	52
4.6	Related Work of GigaE PM	53
4.7	Conclusions of This Chapter	55
5	Hardware Independent Designs	57
5.1	Design Objectives of Cluster Communication	58
5.2	TCP/IP Protocol Processing Overhead	58
5.2.1	Performance in Protocol Layers	59
5.2.2	TCP/IP Protocol Processing Overhead Analysis	60
5.3	A Protocol Handling Scheme Design	62
5.3.1	Protocol Handling for Cluster Computing	62
5.3.2	A Light-weight Reliable Communication Protocol	62
5.3.3	The Interrupt Reaping Technique	63
5.3.4	Cluster Communication and Existing Network Protocols	64
5.3.5	Interrupt Reaping and Existing Network Protocols	64
5.4	Implementation of PM/Ethernet	66
5.4.1	PM/Ethernet Architecture	66
5.4.2	Implementation on Linux	66
5.4.3	Implementation of Interrupt Reaping	68
5.4.4	Multi processes and SMP Support	69
5.5	Evaluation	69
5.5.1	Basic Application Level Communication Performance	69
5.5.2	NAS Parallel Benchmarks (NPB)	70
5.5.3	The Effect of the Interrupt Reaping Technique	72
5.5.4	PM/Ethernet Protocol Processing Cost Analysis	74
5.6	Related Work of PM/Ethernet	75
5.7	Conclusions of This Chapter	75
6	Software Techniques Using Multiple NICs	77
6.1	Network Trunking Design	78
6.1.1	Characteristics of Ethernet Switch	78

6.1.2	The Beowulf Channel Bonding Technique	78
6.1.3	Proposing the Network Trunking Technique	79
6.2	Network Trunking Facility Implementation	80
6.3	Evaluation of Network Trunking	82
6.3.1	PM level Communication Performance	82
6.4	Conclusions of This Chapter	84
7	Comparison of Application Performance	86
7.1	Cluster of Clusters	87
7.2	PMv2 Design	88
7.2.1	<i>COC</i> and SMP Cluster Support	88
7.2.2	Communication Architecture for Multiple Networks	89
7.2.3	Support of Hardware Specific Communication	91
7.2.4	PMv2 APIs	91
7.2.5	Implementation Techniques for PM Devices	93
7.3	PMv2 Implementation	93
7.3.1	Overview of the PMv2 Implementation	93
7.3.2	SCore Version 3 and MPICH/SCore on PMv2	95
7.4	Evaluation on RWC SCore Cluster I	96
7.4.1	Basic Communication Performance on PM/Myrinet, PM/Ethernet	97
7.4.2	PM/Shmem Round Trip Time and PM/Composite Overhead	99
7.4.3	Basic MPI Communication Performance on PM/Myrinet, PM/Ethernet	101
7.4.4	NAS Parallel Benchmark Results	102
7.4.5	Application Performance on an SMP Cluster	105
7.4.6	Summary of Results on SCore Cluster I	106
7.5	Evaluation on RWC SCore Cluster II	109
7.5.1	Basic PM Communication Performance	109
7.5.2	Application Performance on RWC SCore Cluster II	110
7.5.3	Summary of Results on SCore Cluster II	111
7.6	Evaluation on RWC PC Cluster II	113

7.6.1	Summary of Results on PC Cluster II	114
7.7	Conclusions of This Chapter	117
8	Conclusion	118
	Bibliography	119
	List of Publications by the Author	131
	Appendix	137
A	Performance and Costs of Commodity Hardware	137
A.1	CPU Performance and Memory Performance	137
A.2	Hardware Interrupt Costs	138
A.3	I/O Bus Performance	138
B	GigaE PM Protocol	140
C	NAS Parallel Benchmarks	143
D	NPB Results on RWC SCore Cluster I	145
E	NPB Results on RWC PC Cluster II	150

List of Figures

3.1	Original SCore Cluster System Software	26
3.2	Virtual Networks	27
3.3	RWC PC Cluster II	31
3.4	RWC SCore Cluster I	32
3.5	RWC SCore Cluster II	33
3.6	Amber Version 5 Prowat Benchmark Results	35
4.1	Data and Control Flow in TCP/IP Handling	39
4.2	Host CPU Polling effects on PCI DMA transfer: Host to NIC	46
4.3	Host CPU Polling effects on PCI DMA transfer: NIC to Host	46
4.4	Descriptors, Host, and NIC in GigaE PM	48
4.5	GigaE PM Bandwidth	52
4.6	GigaE PM Round Trip Time	53
4.7	CG Class S on GigaE PM	54
4.8	IS Class S on GigaE PM	55
5.1	TCP/IP Protocol Processing in Unix	59
5.2	The Interrupt Reaping Technique	65
5.3	The PM/Ethernet Architecture	67
5.4	Application Level Bandwidth	70
5.5	Application Level Round Trip Time	71
5.6	NPB IS (Class A) on PM/Ethernet	72
5.7	NPB LU (Class A) on an SMP Cluster	73
6.1	The Network Trunking Architecture	81

6.2	Communication Bandwidth on the Digital Tulip	84
6.3	Communication Bandwidth on the Intel EEPROM100	85
6.4	Communication Bandwidth on the 3Com 3C905B	85
7.1	Example of a Cluster of Clusters	87
7.2	Example of a Cluster of Clusters	88
7.3	Protocol Stacks on the UNIX Operating System	90
7.4	PMv2 Architecture	94
7.5	SCore3 Architecture	96
7.6	Communication Bandwidth on PMv2 (100BaseT)	98
7.7	Communication Bandwidth on PMv2 (Gigabit Ethernet, Myrinet)	99
7.8	Communication Round Trip Time on PMv2	100
7.9	MPI Communication Bandwidth on PMv2 (100BaseT)	101
7.10	MPI Communication Bandwidth on PMv2 (Gigabit Ethernet, Myrinet)	102
7.11	MPI Communication Round Trip Time on PMv2	103
7.12	IS CLASS A on SCore Cluster I	104
7.13	CG CLASS A on SCore Cluster I	105
7.14	LU CLASS A on SCore Cluster I	106
7.15	BT CLASS A on SCore Cluster I	107
7.16	IS CLASS A using SMP on SCore Cluster I	107
7.17	LU CLASS A using SMP on SCore Cluster I	108
7.18	Communication Round Trip Time on SCore Cluster II	110
7.19	Communication Bandwidth on SCore Cluster II	111
7.20	NAS Parallel Benchmarks, Class B, 16 nodes, on SCore Cluster II	112
7.21	EP CLASS B on PC Cluster II	114
7.22	BT CLASS B on PC Cluster II	115
7.23	LU CLASS B on PC Cluster II	115
7.24	CG CLASS B on PC Cluster II	116
7.25	IS CLASS B on PC Cluster II	116
D.1	NPB BT CLASS A on RWC SCore Cluster I	145

D.2	NPB CG CLASS A on RWC SCore Cluster I	146
D.3	NPB EP CLASS A on RWC SCore Cluster I	146
D.4	NPB IS CLASS A on RWC SCore Cluster I	147
D.5	NPB LU CLASS A on RWC SCore Cluster I	147
D.6	NPB MG CLASS A on RWC SCore Cluster I	148
D.7	NPB FT CLASS A on RWC SCore Cluster I	148
D.8	NPB SP CLASS A on RWC SCore Cluster I	149
E.1	NPB BT CLASS B on RWC PC Cluster II	150
E.2	NPB CG CLASS B on RWC PC Cluster II	151
E.3	NPB EP CLASS B on RWC PC Cluster II	151
E.4	NPB IS CLASS B on RWC PC Cluster II	152
E.5	NPB LU CLASS B on RWC PC Cluster II	152
E.6	NPB MG CLASS B on RWC PC Cluster II	153
E.7	NPB FT CLASS B on RWC PC Cluster II	153
E.8	NPB SP CLASS B on RWC PC Cluster II	154

List of Tables

2.1	History of Commodity Hardware	8
2.2	Examples of Massively Parallel Computers	13
2.3	Dedicated Cluster Networks	17
2.4	Examples of Cluster Projects Using Dedicated Cluster Networks	17
2.5	Examples of Commercial Cluster System	20
2.6	Summary of Parallel Computing	23
3.1	Original APIs on PM	28
3.2	RWC PC Cluster II Specifications	31
3.3	RWC SCORE Cluster I Specifications	32
3.4	RWC SCORE Cluster II Specifications	34
4.1	Data Transfer Cost Between the Host and NIC Memories and Interrupt / System Call Processing	38
4.2	Access Cost by Descriptors Location	44
4.3	Access Cost by a Flag Location	45
4.4	Trigger Cost From the NIC to the Host	45
4.5	Comparison of Cost in an N Word Message	50
4.6	Comparison of Memory Usage in an N node Cluster	50
4.7	Evaluation Environment for GigaE PM	51
5.1	Evaluation Environment for TCP/IP	59
5.2	Round Trip Time and Bandwidth in Protocols	60
5.3	TCP/IP Overhead	62
5.4	Number of Interrupts on IS	73

5.5	Interrupt Reaping on Other Platforms	74
5.6	Comparison of Protocol Processing Cost Analysis on 1/2 of the Round Trip Time	74
6.1	Evaluation Environment for Network Trunking	83
6.2	Round Trip Time of Network Trunking (μ sec)	83
7.1	An Example of a Destination PM Device Table on PM/Composite	89
7.2	Protocol Processing Requirements on Each Network	90
7.3	APIs on PMv2	92
7.4	Measurement Environments on RWC SCORE Cluster I	97
7.5	Communication Round Trip Time on PMv2	99
7.6	Communication Round Trip Time on PM/Shmem	100
7.7	MPI Communication Round Trip Time	103
7.8	Measurement Environment on RWC SCORE Cluster II	109
7.9	Measurement Environment on RWC PC Cluster II	113
A.1	SPEC95, System Call Cost and Memory Copy Performance	137
A.2	Hardware Interrupt Cost	138
A.3	PCI DMA Performance	139
C.1	NAS Parallel Benchmarks and Dominant Message Characteristics on Class A, 16 Node programs	144

Chapter 1

Introduction

This thesis proposes design techniques to achieve high-performance communication for parallel computers using a commodity network implemented at the software level. The proposed techniques are based on the balance among processor performance, system bus performance, memory performance and I/O bus performance in the communication scheme for a parallel computer. Moreover, hardware and software techniques should be combined effectively in order to design and achieve the goal of high-performance communication.

1.1 The Need for Parallel Processing and Cluster Systems

The personal computer (PC) has been augmented with new information processing techniques such as multimedia processing. The advance of PCs is based on the remarkable development of microprocessing technology and the improvements of cost performance in recent years.

The spread of PCs pushes the advances in intranet and Internet technologies, such as the *World Wide Web* (WWW) and electronic commerce, and vice versa. This, in turn, increases the speed of networks, e.g., 10/100/1000 Mbps in a local area network, and reduces hardware costs dramatically. In intranets and on the Internet, there are many applications that process huge volumes of transactions, such as WWW services, mail services, news services, streaming-video services, and data mining.

In addition, large-scale computation power is required for simulation and data-mining. For example, a new information technology, called *Bioinformatics*, has emerged and holds promise for understanding disease mechanisms, as well as designing drugs and macromolecular materials, and so on. This technology needs molecular dynamics applications and data-mining of the human genome that searches huge volumes of data and requires high performance computing [Aki99]. Another example is the *Earth Simulation System*, represented by the “Earth Simulator” [ETH], for understanding climate and ocean systems, and predicting floods, global warming, and weather disasters. These simulations require huge amounts of computation power to run climate models, weather models and fluid dynamics problems.

To process such huge data tasks, the demand for parallel computers has been increased. Previously, such a parallel computers could not be built without dedicated hardware, because there were neither standard I/O buses nor standard Gigabit class networks. Recently, on account of improvements in computing performance in the PC field, and improvements in the speed of computer networks, a cost-effective parallel processing system can be built by combining commodity hardware. Such systems are called cluster systems.

In the early 1990s, some researchers had tried to build clusters using commodity net-

works, such as 10/100Mbps Ethernet and ATM/LAN. However, performance was limited compared to that of commercial parallel computers due to the network hardware limitations. In the middle of the 1990s, several research projects started to build clusters with a dedicated cluster network. These projects showed that a network of PCs was capable of becoming a supercomputer. Through these research efforts, clusters with dedicated cluster networks have already become very popular and important in building high performance commercial super-computers.

With the appearance of Gigabit Ethernet, Gigabit class communication has become possible on commodity networks. This fact shows that the performance of commodity networks has caught up with that of dedicated cluster networks. Using Gigabit Ethernet, a high performance cluster system, with application performance comparable to that of dedicated networks, can be built in a LAN environment.

1.2 Issues in Cluster Systems Using Commodity Networks

Usually, the TCP/IP Internet protocol is used on cluster systems using commodity networks. The communication performance of the TCP/IP protocol on Gigabit Ethernet achieves only 45MB/s in total bandwidth, even though the physical link performance is 125MB/s[FO00]. Cluster systems using the TCP/IP protocol cannot ensure optimum performance for either present or future network hardware. A communication facility to enable high-performance communication is needed in order to build parallel computers using Gigabit class commodity networks.

Since Gigabit Ethernet does not guarantee message transfer at the hardware level, a reliable communication protocol must be implemented. Moreover, the size of the maximum transfer unit (MTU) of Gigabit Ethernet is 1514 bytes. This fact shows that a reliable protocol processing of each packet must be finished within 12 μ s at a 125 MB/s transfer rate, and within 15 μ s at a 100 MB/s transfer rate. Thus a light-weight reliable protocol and design methods to minimize the overhead become very important and are issues to be solved on communication facilities on Gigabit Ethernet.

1.3 The Purposes of This Thesis

The first purpose of this thesis is to propose implementation designs which ensure the maximum communication performance on cluster systems using Gigabit Ethernet. The second is to show that a communication facility using Gigabit Ethernet can achieve application performance comparable to that of dedicated-cluster network communication.

In order to realize the first purpose, the following designs to improve communication performance are proposed:

“Hardware dependent designs”:

Designs to minimize the information exchange cost between the host and network interface card (NIC) are proposed. There are significant design choices: where should a reliable protocol be processed, in the host CPU or in the NIC CPU; and where should the data structures for managing the send and receive buffers and triggering be allocated, in the host or in the NIC memory. Since those structures are shared resources, the cost between host and NIC information exchange is crucial in performance.

In this approach, the cost of TCP/IP processing is analyzed using a model of information exchange for TCP/IP, and for this, the following method and protocol have been developed:

- A design method for minimizing the aforementioned information exchange cost
- A reliable light-weight communication protocol called “GoBack-N for PM”

The GigaE PM communication facility has been implemented, and evaluated in terms of its communication and application performance.

“Hardware independent designs”:

Designs to reduce the processing overhead of existing protocol stacks are proposed. To propose appropriate methods, the cost of the TCP/IP communication protocol is analyzed. Then, a new protocol-processing scheme based on the results of a detailed cost analysis of TCP/IP processing is proposed.

Using this scheme, PM/Ethernet has been implemented and its performance evaluated. In addition, the *Network Trunking* technique to improve bandwidth performance using

multiple NICs has been developed, and evaluated.

To achieve the second purpose, application performance on Gigabit Ethernet is compared with that on a dedicated cluster network using a communication facility which supports both dedicated cluster networks and commodity networks. The NAS parallel benchmarks [NPB] are used to measure application performance, because they include effective algorithms for computational fluid dynamics (CFD) applications and some of these reflect the communication performance quite well.

1.4 Overview of This Thesis

Chapter 2 describes the background and needs of parallel processing and introduces some studies of cluster systems, especially those involving communication facilities. Chapter 3 gives an overview of the Real World Computing (RWC) project and the position of this thesis in the project. Chapter 3 also discusses the objectives of this thesis and the SCore software that is the implementation and evaluation platform used in this thesis. The "hardware dependent designs" are described in Chapter 4; the "hardware independent designs" in Chapter 5; and, the "software techniques using multiple NICs" in Chapter 6. Finally, the "comparison of application performance with a dedicated cluster network" is described in Chapter 7.

1.5 Contributions

The original contributions of this thesis are the following software designs needed to make a cluster system with commodity networks and to achieve application performance comparable to that of dedicated-cluster network communication:

- Hardware dependent designs: Designs to minimize information exchange cost between the host and network interface card (NIC) are proposed. GigaE PM is implemented using these designs. The evaluation results show that about half of the information exchange cost of TCP/IP is eliminated.

- Hardware independent designs using existing NICs: Designs needed to reduce the processing overhead of existing protocol stacks are proposed. PM/Ethernet is implemented using these designs. The evaluation results show that two thirds of the protocol-processing overhead of TCP/IP is eliminated.
- Software techniques using multiple NICs: Designs needed to achieve high bandwidth using multiple NICs are proposed.

Chapter 2

Background

Cluster systems are not discussed without considering the histories of commodity hardware (such as processors and networks) and parallel computers. This chapter describes the history of commodity hardware and parallel computing and explains the background of cluster systems using commodity networks, including surveys of their cluster related projects, especially those involving software techniques used to realize high-performance parallel computation.

2.1 History of Commodity Hardware

Table 2.1: History of Commodity Hardware

Year	Microprocessors	Max Clock	I/O Bus on PC	Ethernet
1971.	4004	108 KHz		
1972.	8008	108 KHz		
1974.	8080	2 MHz		
1978.	8086	10 MHz		
1981.			XT 2.4 MB/s	10 Mbps
1982.	80286	12.5 MHz		
1984.			ISA 8 MB/s	
1985.	Intel 386DX	33 MHz		
1987.			EISA 33 MB/s	
1989.	Intel 486DX	50 MHz		
1992.			PCI 132 MB/s	
1993.	Pentium	166 MHz	(32bit 33MHz)	100 Mbps
1995.	Pentium PRO	200 MHz		1 Gbps
1997.	Pentium II	450 MHz		
1999 -.	Pentium III	1.13 GHz	PCI 533 MB/s	
2000 -.	Pentium 4	1.5 GHz	(64bit 66MHz)	

Microprocessors

As shown in Table 2.1[HIN, HCP, PCI, SPC, SIP, SIS, SET], the microprocessor was invented to make an electronic calculator in 1971. Since then, it has been used in embedded systems, industrial process controls, telephone exchange switches, and so on. In the 1980's, there were two new uses for the microprocessor technology: the personal computer, shortly to be known as the PC, and the engineering workstation(WS). At the beginning of PC market development, there were many kinds of PCs, each using different microprocessor architectures, such as Motorola 6800, Intel 8086, Zilog Z80, and so on. PCs were used for word processing, spread sheets, small transaction processing, and so on. As the PC market grew, competition intensified. Since IBM introduced the PC/XT and PC/AT using the Intel 16 bit 8086 microprocessor, known as the x86 architecture, many vendors have duplicated the specifications of IBM PCs. Some vendors, such as AMD and Cyrix, have developed microprocessors, pin-compatible to the Intel 8086 architecture. Now, the word

PC refers to a computer using the Intel x86 architecture. Severe competition reduced the life cycle of microprocessors. However, because the Intel microprocessor 8086 and its successors did not have floating point units on one processor chip until late 1980s. The PC was limited to personal use. There was an 8087 floating point co-processor, but it was too expensive for personal use.

Since reduced instruction set computer (RISC) architecture[PD80] was introduced in the beginning of the 1980s, RISC microprocessors with floating point and memory management units, such as the SPARC, Alpha and MIPS chips, have been developed to build engineering workstations. The target markets of engineering workstations were computer-aided design, computer graphics, scientific calculation, and so on. Due to a smaller market size than that of the PC, the cost of the workstation could not be reduced.

In the early 1990s, Intel introduced the new Pentium architecture which increased processor clock speed, supported a 64bit system bus and a multi-processor, upper compatible to the x86 architecture. At that time, the x86 architecture was close to RISC-based architectures in terms of functionalities. The performance, however, was slower than the RISC-based architectures because of a lack of processor clock speed.

In the middle of the 1990s, the Pentium architecture's performance was dramatically improved (Appendix A.1) because more processor performance was required for multimedia processing, graphical user interfaces, and so on. Especially, the cost performance of PC was getting close to that of an engineering workstation using a RISC-based architecture, because the x86 processor was less expensive than a RISC-based processor as the result of severe competition and a larger market than that of the engineering workstation. Thus, after the middle of the 1990s, PCs were not only just for word processing, but were also used to the replace engineering workstations.

The I/O Bus

In early computer systems, there was no standard I/O bus, so PC manufacturers used proprietary I/O busses. In PCs, each PC manufacturer developed its own I/O bus, such as the IBM PC/XT bus. The PC/XT bus was completely under the Intel 8088 processor's direct control, and its addressing was limited to a width of 8-bits. In spite of many

improvements to the specification, bus speed was also limited to match to the processor. The Intel 8088 on PC/XT was an eight bit, 4.77 MHz processor. Thus the XT bus, which required two clock cycles for data transfer, was limited to 2.38 MB/s.

The Industry Standard Architecture (ISA) bus, as part of the PC/XT, was released in 1981. The ISA bus was used by the Intel 80286, and was designed to run at 8MHz with a full 16-bit data bus. With the spread of the ISA bus, It became one of the industry's standard I/O busses. Still limited by its two-clock-cycle data transfer, the ISA bus can reach speeds of only 8MB/s. Extended ISA (EISA) was developed to improve ISA bandwidth by a data bus expansion to 32 bits. However 33MB/s of EISA data transfer was not enough to satisfy the demands of the processing speed of 32 bit processors, such as the Intel Pentium processor.

The appearance of the PCI bus in 1992 expanded data transfer speeds to 132MB/s (32bits, 33MHz) matching current processor speeds. It is very likely that ISA slots will soon disappear from new PCs altogether. The PCI specification continues to improve, for example, with 64bit, 66MHz expansion. The 64bit, 66MHz PCI achieves 533MB/s data transfer¹.

Commodity Networks

In the early 1980s, 10Mbps Ethernet LANs began to be used, early implementations of Ethernet LANs employed thick coaxial cable (yellow cable), defined by the 10Base5 standard. Network software, such as telnet and network file systems, were developed and used on computers connected with Ethernet. Unfortunately, the yellow cable was difficult to work with, because a miss-operation caused the network to crash when nodes were added or removed. In the middle of 1980s, 4Mbps Token-Ring LAN was introduced for the original IBM Personal Computer. The Token-Ring LAN was implemented on shielded twisted pair (STP) cable which was easy to use, so users of Token-Ring LAN increased rapidly.

In the early 1990s, with the increasing number of computers connected to LANs, a 10Mbps network was not fast enough to be shared by numbers of computers. After 100Mbps

¹Some PCI bus performance is shown in Appendix A.2.

FDDI was introduced, FDDI usage expanded as a backbone LAN because FDDI had higher bandwidth and could be used over longer distances than other LAN technologies. Moreover, FDDI has dual-ring LAN technology for redundancy. At that time, a new Ethernet LAN, based upon UTP (un-shielded twisted pair) and defined by the 10BaseT standard, was again much easier to use than 10Base5. The other feature of 10BaseT is a hub and spoke architecture. The various types of data equipment are all connected to a central point called a Multi-port Repeater or Hub by UTP cable. This reduces the problem of adding or removing nodes. Due to these features, 10BaseT took off in the market place.

The networking of computers has been spreading rapidly since the appearance of 10BaseT with increasing demands for network computing, such as on-line transactions, data-base and file sharing servers, and especially the spread of intranet and Internet technologies represented by growth in the use of the *WWW*. Thus, a number of vendors supply the same equipment, and both technological innovation and price point competition between vendors become intense. As the result of competition, the cost performance of Ethernet equipment continues to increase. For the LAN market place, 10BaseT was too slow to share among a number of hosts. This led to the development of 100BaseT (a twisted cable version of Fast Ethernet) including 100BaseF (a fiber version) in 1993. Another technology innovation was Ethernet switch technology to improve point-to-point bandwidth individually. With these technology innovations for Ethernet, the use of FDDI gradually began to decrease.

In the middle of the 1990s, with the increasing needs of multimedia processing, such as video-on-demand, asynchronous transfer mode (ATM) LAN was introduced. The 155Mbps ATM/LAN is a technology which supports QOS (quality of service), and is based on broadband ISDN (B-ISDN). So, users can use an ATM network not only as a LAN environment but also as a WAN environment without converting packet frames. However, as 155Mbps ATM/LAN network interface cards (NICs) and switches were very expensive at that time, the use of ATM/LAN did not increase. So a cost-effective version of ATM25 (a 25Mbps version) was proposed.

At the same time, Fast Ethernet has achieved remarkable growth in the LAN market place because of having the best interoperability with 10Mbps Ethernet. With the growth of the Ethernet market place, the cost of 10BaseT and 100BaseT Ethernet has dropped

dramatically, and no other network could match the cost of Ethernet. 100BaseT is not the end of the road. The speed of an Ethernet network increased to 1Gbps (Gigabit Ethernet) in 1995. Vendors are now demonstrating a proposal for 10Gbps Ethernet for a metropolitan network.

2.2 The History of Massively Parallel Computers

Since Cray, Inc. introduced the CRAY-1 vector parallel computer in 1976, the first CRAY-1 was installed at Los Alamos National Laboratory in 1976 for 8.8 million dollars. It boasted a world-record speed of 160 Mflops (million floating-point operations per second). Cray and other computer vendors, such as NEC and Fujitsu, have developed vector parallel computers whose processing units are dedicated and can not be used for other purposes. The development of new systems involves not only processors, but also the development of operating systems, compilers, and programming environments. This results in an expensive supercomputer and a long development cycle. The CRAY-2 achieved 1.9 Gflops in 1985, and the CRAY C90, 16 Gflops in 1991. In the 1970s and 1980s, there were no alternatives to achieve a high performance computing environment. Thus, supercomputers were popular in the high performance computing field.

After RISC-based architectures were introduced in the late 1980s as described in section 2.1, some computer vendors, such as Thinking Machine, Intel, and Cray, thought that thousands or millions of microprocessors connected by a high speed network would enable the building of a high performance computing environment which would replace vector computers for some applications. Because neither standard high performance I/O busses nor networks were available in the late 1980s, computer vendors started to develop dedicated high performance networks, which constitute parallel computers using microprocessors, from scratch. Those computers were using RISC-based architectures because the performance of RISC-based architectures was higher than that of other microprocessors including the x86 architecture in the late 1980. Such a parallel computer was referred to as a massively parallel computer (MPP). The first MPPs were announced in the beginning of the 1990s, as shown in Table 2.2. At that time, the performance of the MPP was about

ten times higher than that of vector computers, such as the CRAY C90.

Unlike a vector computer involving the development of a dedicated processor, the cost of development was reduced in the high performance computing market. However, it still involved the development of high performance network and system software.

Though the length of the development cycle was reduced in the high performance computing market, the development cycle was reduced dramatically in the PC market. Every half year, the microprocessor's clock speed was improved in 1990s. This means that a MPP cannot catch up to the latest microprocessor technology. Most vendors withdrew from the market in the latter half of the 1990s.

Table 2.2: Examples of Massively Parallel Computers

Year	Machine	Vendor	Processor	Network Bandwidth per Processor	Peak Performance
1992.	CM5	Thinking Machines	SuperSparc 32MHz	20 MB/s	135 Gflops (1056PEs)
1992.	Paragon XP/S	Intel	Intel i860 XP 50MHz	200 MB/s	184 Gflops (3680PEs)
1993.	T3D	CRAY	Alpha 21064 200MHz	300 MB/s	307 Gflops (2048PEs)
1995.	T3E	CRAY	Alpha 21164 300MHz	500 MB/s	1229 Gflops (2048PEs)

Note: Peak Performance [RTO]

2.3 Standardization of the Parallel Application Programming Environment

MPP, which is described in section 2.2, provided its own special communication facility for users to write parallel applications, for example, Paragon provided the NX message passing library, CM5, the CM message-passing communications library, CMMD, and T3D, the shmem library. So, users had to write parallel applications using proprietary libraries. With the increase of the kinds of MPPs available, the portability of parallel applications between different computers was lost. Moreover, some applications developed could not

be used because some vendors withdrew from the market. Thus, the portability of the applications to other MPPs became a very important issue to be solved.

To realize the needed portability of applications, PVM[PVM] and MPI[MPIa] have been proposed. A user program written using PVM or MPI can run on other machine platforms without modifications. They are based on the message passing model. Message passing is a paradigm used widely on certain classes of parallel machines, especially those with distributed memory. The main advantages of establishing a message-passing standard are portability and ease-of-use. In a distributed memory communication environment in which the higher level routines and/or abstractions are built upon lower level message passing routines, the benefits of standardization are particularly apparent.

PVM

PVM[PVM] is a message passing library which is designed to be run on heterogeneous networks, and which has good interoperability between different hosts, because the PVM model is based on the virtual machine concept. PVM provides a set of dynamic resource manager and process control functions, and includes runtime environments.

MPI

MPI[MPIa] is a message passing library specification which is designed to run faster within a large multiprocessor. MPI has many point-to-point and collective communication options. Moreover, it has the ability to specify a logical communication topology. There are a lot of MPI implementations, such as MPICH[MPIb], and MPI-LAM[LAM] and so on. These environments can be used not only on MPPs but also on PCs or workstations connected by a LAN.

2.4 A History of Cluster Computing

In addition to MPP technology, many kinds of local area network (LAN), such as Ethernet, FDDI, ATM/LAN, Token-Ring, have been developed as described in section 2.1. With the spread of the LAN environment in the 1980s, some researchers started to make distributed operating systems, such as Accent[RR81], Mach[JR86], V-system[TLC85], Sprite[OCD⁺88],

and so on. Some of them, such as Mach and V-system, had a mechanism allowing users to run parallel jobs, and making them suitable as commercial or research environments. However, in order to use the operating systems, users had to install new operating systems and write parallel programs using special APIs of the operating system.

2.4.1 Cluster Computing Using Commodity Networks in the Early 1990s

In the early 1990s, some free Unix-based operating systems, such as Free BSD and Linux, made it possible to use a Unix-based operating system on PCs with LAN environments. Some researchers had tried to build clusters using commodity hardware, such as existing PCs, workstations, and 10/100Mbps Ethernet or ATM/LAN. Examples include U-NET[BBVvE97] and the Beowulf[BEO] project.

The Beowulf Project

The Beowulf project[BEO] proposed a parallel computer using existing PCs running on Linux, a commodity network, the TCP/IP protocol, and MPI[MPIa] (or PVM[PVM]), as described in section 2.3, on top of the TCP/IP protocol. All of the components were composed of existing parts. The Beowulf project developed a number of Ethernet device drivers on Linux in order to spread their style of super-computing. Beowulf has also proposed the *Channel Bonding* technique [T. 95] to achieve high-bandwidth communication using multiple 10Mbps Ethernet NICs.

U-NET

The system call overhead in computer systems in the first half of the 1990s was very expensive, for example, that of the 5.5 μs SuperSPARC 75MHz system. In order to eliminate this overhead, U-NET[BBVvE97] used user-level communication on commodity networks, such as Fast Ethernet and ATM in 1994. U-NET uses existing PCs running on Linux (or Windows NT) and commodity networks. A user-level communication facility handles network interface hardware directly by means of the user program without using hardware interrupts. U-Net/MM[MW97] also supports *Zero-Copy* communication on ATM/LAN and Fast Ethernet. The *Zero-Copy* communication is a technique to allow applications to

transfer data directly between virtual memory address spaces over networks.

These projects were different as to whether an existing network protocol was used or not. However, performance was limited compared with MPPs, due to the network hardware bandwidth limitations.

2.4.2 Cluster Computing Using Dedicated Cluster Networks

Since 1992, PCs and workstations began to have a standard I/O bus, such as the PCI bus, capable of more than 100MB/s transfer throughput. The PCI bus can utilize a Gigabit class network. Several kinds of Gigabit class networks on the PCI bus have been developed including Myrinet [N. 95, MYR] and Memory Channel [MEM], and SCI [SCI]. (Table 2.3). These networks are briefly described as follows:

- **Myrinet** [N. 95, MYR]

Myrinet is a 1.28 Gbps (160 MB/s) full duplex interconnection network supplied by Myricom. Myrinet uses low latency cut-through routing switches and supports reliable message transfer at the hardware level. A large Myrinet network can be built with combinations of a number of Myrinet switches. The Myrinet network interface card has an on-board programmable processor and local memory, which a host processor can access. So the user can add special functionality using the NIC processor.

- **Memory Channel** [MEM]

Memory Channel is a 133 MB/s dedicated cluster interconnect system that provides virtual shared memory among nodes. The Memory Channel supports reliable message transfer, and uses a memory mapped connection realized in hardware, through a global Memory Channel address space. A page of virtual memory in one node is connected by hardware to a page of virtual memory in another node.

- **SCI** [SCI, DOL]

Scalable Coherent Interface (SCI), an ANSI/IEEE 1596-1992 standard defines a point

to point interface and a set of packet protocols. An SCI interface has two unidirectional links that operate concurrently. The SCI protocols support shared memory by encapsulating bus requests and responses into SCI request and response packets. Packet-based handshake protocols guarantee reliable data delivery. A set of cache coherence protocols is defined to maintain cache coherence in a shared memory system.

Table 2.3: Dedicated Cluster Networks

Network	Vendor	Link Bandwidth	I/O bus
Myrinet	Myricom	160 MB/s	PCI, S-Bus
Memory Channel 2	Compaq	133 MB/s	PCI
SCI	Dolphine	200 MB/s	S-Bus
SCI	Dolphine	400 MB/s	PCI

Table 2.4: Examples of Cluster Projects Using Dedicated Cluster Networks

Projects	Organization	Processor Architecture	Cluster Network	Communication Facilities
NOW[NOW]	UCB	Sparc	Myrinet	AM, AM-II
HPVM[HPV]	NCSA	Intel x86	Myrinet	FM
C-Plant[CPL]	Sandia National Lab	Alpha	Myrinet	(self-made)
SHRIMP[SHR]	Princeton University	Intel x86	Myrinet	VMMC, VMMC-II
BIP[PT98]	RESAM Laboratory	Intel x86, Alpha, PowerPC	Myrinet	BIP
RWC[RWC]	RWCP	Sparc, Intel x86, Alpha	Myrinet	PM

A number of cluster-research projects were started using dedicated cluster networks, as shown in Table 2.4. They developed dedicated communication facilities in order to realize high-performance communication as shown below:

The Berkeley NOW project

The Berkeley NOW project [NOW] used Sun SPARC systems, and has developed high-performance communication facilities called Active Message (AM)[T. 92, T. 94] and AM-

II[CMC97], a global operating system called GLUnix, and a distributed file server called xFS. The Multi-protocol AM[LMC97] is able to support shared memory communication and Myrinet communication to implement a switch interface to both communications. AM and AM-II use a remote procedure model where a message includes both a control and data. So, every network operation involves a round trip message transfer. AM and AM-II support flow control in the receive buffer. They do not support virtual networks in order to support the running of multiple processes.

The HPVM Project

The goal of High Performance Virtual Machines (HPVM) [HPV] is to increase the accessibility and delivered performance of distributed computational resources for high performance computing applications. HPVM supports number of APIs: Fast Messages (FM)[Sco95], MPI-1, SHMEM, and Global Arrays. Fast Messages (FM) is a low-level messaging layer designed to deliver underlying network's hardware performance to the application, even for small messages. FM uses a send and receive message model with FIFO message delivery, and a message of FM contains a function pointer and data. The data is processed at receiver node using the function pointer. FM supports both flow control of receive buffer and virtual network.

C-Plant

The Computational Plant (C-Plant) project [CPL] at Sandia National Laboratories is developing a large-scale, massively parallel computing resource from a cluster of commodity computing and networking components. C-Plant uses the partition model. All of the nodes on the high-speed network, such as Myrinet, are treated by the system as a single pool of nodes. Administrators can divide the machine into several functional partitions: service, compute, disk I/O, and network I/O. All of these partitions run Linux, and kernel modules are used to adapt the operating system to the functionality of the partition. A communication facility with special firmware was developed on the Myrinet network.

The SHRIMP Project

The goals of the SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project[SHR] is to study how to build such a system to deliver performance competitive with or better than commercial multi-computer servers. The SHRIMP research topics are protected user-level communication, efficient message passing, shared virtual memory, and so on. Virtual memory-mapped communication (VMMC)[DBLP97] was developed in order to realize low latency and high bandwidth as a part of protected user-level communication on the SHRIMP project. VMMC uses a *Zero-Copy* technique over the Myrinet network. VMMC-2[DBC⁺97] was designed to overcome the deficiencies of VMMC model. VMMC-2 extends VMMC with the following mechanisms: a user-managed TLB mechanism for address translation and a reliable communication protocol at the data link layer.

BIP

BIP (Basic Interface for Parallelism)[PT98] is communication facility designed and implemented on the Myrinet network for single user systems. BIP messages are implemented for cluster of workstations connected by Myrinet boards with the LANai processor. BIP also supports *Zero-Copy* communication. The implementation consists of a user-level library associated with a custom MCP that will run on the Myrinet board. BIP-SMP[GPT99] is also able to support shared memory communication and Myrinet communication to implement a switch interface to both types of communication. BIP does not support flow control of the receive buffer nor that of the virtual network.

RWC

Since 1995, the Real World Computing Partnership (RWCP) has been researching a cluster system software called the SCore Cluster System software[HIK⁺93, HIS⁺93, HTI98] to realize a seamless parallel and distributed system. The SCore Cluster System software consists of a communication facility on Myrinet called PM[THI96, THIS97, TOHI98], an MPI implemented on PM called MPICH-PM/CLUMP[TOT⁺99], a global operating system called SCore-D [HIK⁺95, HYI⁺95, HIN⁺95, HTI⁺96, HTI98], a software distributed shared memory system called SCASH[HIH⁺00], and a multi-threaded programming language called MPC++[IHK⁺94, IHS⁺95, IHT⁺96, TISY]. PM uses a send and receive

message model and remote memory model. PM also supports both flow control of the receive buffer and of the virtual network. Chapter 3 describes the details of the SCore Cluster System software.

These research projects showed that a network of workstations was capable of becoming a supercomputer. Through these research results, cluster computing with dedicated cluster networks has already become very popular and important in building high performance commercial super-computers, such as the IBM SP and the Compaq SC series (Table 2.5).

Table 2.5: Examples of Commercial Cluster System

Year	Machine	Vendor	Processor	Network Bandwidth per Processor	Peak Performance
1999.	Alpha SC	Compaq	Alpha 21264 667MHz	210 MB/s	683 Gflops (256PEs)
1999.	RS/6000 SP	IBM	Power 3 375MHz	160 MB/s	3070 Gflops (2048PEs)

Note: Peak Performance [RTO]

With the increasing demands of dedicated cluster networks, lack of standardization of the networks causes portability problems of application programs between the cluster networks. So, in 1997, Compaq, Intel and Microsoft proposed the Virtual Interface Architecture in order to make standard application interfaces on dedicated cluster networks.

VIA[VIA], the Virtual Interface Architecture, specifies an industry-standard architecture for communication within clusters of servers and workstations, and defines user-level communication APIs. VIA is being widely implemented in Gigabit class networks on the Microsoft Windows operating system. The design of VIA is based on U-NET[BBVvE97]. VIA supports connection oriented communication. Reliable communication support is an option according to the VIA specification, version 1.0. However, VIA is not an architecture for users to carry out scientific parallel computation. There are some commercial implementation of VIA, such as Giganet, and research implementations, such as M-VIA[MVI], the Berkeley VIA[BVI].

2.4.3 Cluster Computing Using Commodity Networks in the Late 1990s

As already described in section 2.1, the speed of an Ethernet network has increased to 100Mbps (Fast Ethernet) in 1993, and 1Gbps (Gigabit Ethernet) in 1995. Beowulf-type clusters have been spreading in network technology with clusters consisting of a small number of nodes. Because of the spread of high cost-performance PCs and Ethernet equipment, The GAMMA (Genoa Active Message MACHine) Project has been researching cluster computing on commodity networks.

GAMMA[CC00] runs on clusters of PCs, based on the Intel architecture, connected by Fast Ethernet or Gigabit Ethernet. At first, GAMMA had been implemented on Fast Ethernet, but was not expected to be implemented on Gigabit Ethernet. The core of GAMMA is a custom device driver under Linux, which operates the NIC. The GAMMA driver delivers very low latency, high bandwidth communications using Active Ports, a mechanism derived from Active Messages. The GAMMA driver is able to manage standard IP traffic in addition to GAMMA fast communications. This means that using GAMMA on your LAN will not stop use of the standard UNIX network services. However, GAMMA does not provide reliable communication, so a program fails when a message is lost. Also, the implementation of GAMMA is dependent on NIC device drivers.

2.5 Issues of Cluster Computing on Gigabit Class Commodity Networks

The appearance of Gigabit Ethernet has made Gigabit class communication possible on commodity networks. The TCP/IP protocol is the most popular communication protocol on commodity networks, such as Gigabit Ethernet, so TCP/IP is usually used for cluster systems on these commodity networks. Many vendors and researchers have tried to evaluate and improve TCP/IP performance on Gigabit Ethernet, however the message transfer performance of TCP/IP continue to be only less than 50MB/s[HAM, SCL, SHT⁺99b]. Some vendors proposed the JUMBO frame which expanded the maximum transfer unit (MTU) up to 9000 bytes, however, a special Ethernet switch is required and this loses

interoperability with other Ethernet NICs.

A paper entitled “Communication Performance over a Gigabit Ethernet Network” [FO00] measured TCP/IP communication performance on Gigabit Ethernet and on several versions of the Linux operating system. The communication bandwidth of the TCP/IP Internet protocol on Gigabit Ethernet only achieved 45MB/s using the G-NIC II and the Pentium II 350MHz processor although the physical link performance is 125MB/s. Thus, the TCP/IP protocol can not ensure the maximum communication performance of the network hardware of a Gigabit Ethernet. A communication protocol to ensure the maximum communication performance of the network hardware is needed in order to build parallel computers using Gigabit class commodity networks.

Since Gigabit Ethernet does not guarantee message transfer at the hardware level, a reliable communication protocol must be implemented. Moreover, the size of the MTU of Gigabit Ethernet is 1514 bytes. This fact shows that the reliable protocol processing of each packet must be finished within 12 μs at a 125 MB/s transfer rate, and 15 μs at a 100 MB/s transfer rate. Thus a light-weight reliable protocol and design methods to minimize the overhead are issues to be solved.

2.6 Summary of This Chapter

- Parallel computing is summarized in Table 2.6. The History of cluster systems can be considered as the history of the standardization of hardware and software.
- Cluster systems using PCs and workstations have become popular due to the remarkable progress in microprocessors, I/O buses and high speed cluster networks. A lot of cluster research projects have been carried out.
- The TCP/IP protocol can not ensure the maximum communication performance of the network hardware of a Gigabit Ethernet, one of the next generation of commodity networks.
- A light-weight reliable protocol and design methods to minimize protocol overhead are needed to ensure the maximum communication performance of the network hardware.

Table 2.6: Summary of Parallel Computing

Categories	Hardware				Software	
	Processor	Logic Board	I/O Bus	Network	OS	Network Protocol
Proprietary Vector Parallel Computers	-Ded	-Ded	-Ded	-Ded	-Ded	-Ded
Massively Parallel Computers	+Ext	-Ded	-Ded	-Ded	-Ded	-Ded
Cluster Computing with Dedicated Network	+Ext	+Ext (PC)	+Ext (PCI)	-Ded (Myrinet)	+Ext (Linux)	-Ded
Cluster Computing with Commodity Network	+Ext	+Ext (PC)	+Ext (PCI)	+Ext (Ethernet)	+Ext (Linux)	-Ded +Ext (TCP/IP)

+Ext: Existing, -Ded: Dedicated, (): Ex.

Chapter 3

The RWC Project

This chapter describes an overview of the Real World Computing (RWC) project in which this thesis has done. Especially, the SCore Cluster System software (SCore Software) developed in the RWC project is described, as it is the platform used in this thesis. Hardware platform RWC clusters are also described. These cluster systems were developed in order to demonstrate that the performance of a cluster system was equal to that of a commercially available parallel computer. The SCore Software has been installed and used as a practical parallel processing environment for over two years in several places around the world.

This thesis has done as a part of SCore Software development. So, the goal of a communication facility on a commodity network as discussed in this thesis is actually to realize the SCore software on a commodity network and to demonstrate its effectiveness in practical use.

3.1 The RWC Project

Since 1992, the Real World Computing Partnership (RWCP) has been actively promoting the Real World Computing (RWC) Project, a 10-year plan, under the guidance of the Ministry of International Trade and Industry (MITI), to create an innovative system of information processing technologies capable of handling various types of information in the real world.

One of the main research fields of the RWC project is Distributed Computing Technology. In this field, the Parallel and Distributed System Software Laboratory has been researching a cluster system software called the “SCore Cluster System Software” to realize a “seamless parallel and distributed system” (seamless system)[ISS96] since 1995. The seamless system will become a next-generation parallel and distributed system for the LAN of the future with a transmission throughput ranging from several Gbps to 10 Gbps. A seamless system is a distributed system that enables users to draw the optimal computing power to meet users’ needs from computers on a network in a distributed environment using a high-speed LAN. The seamless system must also support different CPU architectures and different types of networks.

3.2 SCore Cluster System Software

A cluster system consists of a number of nodes which are connected by a network, so when a parallel program is executed, some software is needed to provide an environment which supports start-up function of parallel processes on the cluster nodes. Moreover, when a cluster system is used by multiple users, some resource management, such as global scheduler, which provides automatic dispatching to the unused nodes, time sharing scheduler and so on, should be provided.

Cluster System software is not only software which provides the start-up function of parallel processes, but also software which provides a multi-user environment, job management and the other sophisticated features, such as checkpoint-restart for long-period jobs. In a practical cluster system environment, users only specify the number of nodes and a program, then, cluster system software allocates the number of nodes and executes the user

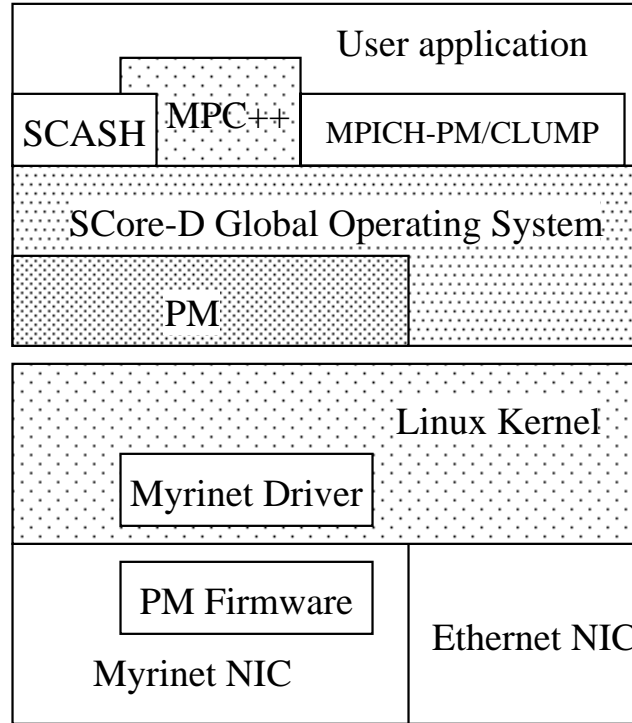


Figure 3.1: Original SCore Cluster System Software

program automatically.

The SCore cluster system software (SCore software) [HIK⁺93, HIS⁺93, HTI98] is a system software designed to realize a seamless system. This thesis has done as a part of the SCore software development.

Figure 3.1 shows the architecture of the original SCore software which does not include software developed in this thesis. The original SCore software consists of a communication facility on Myrinet called PM[THI96, THIS97, TOHI98], an MPI implemented on PM called MPICH-PM/CLUMP[OHT⁺97, OTHI98, TOT⁺99], a global operating system called SCore-D[HTI⁺96, HTI98], a software distributed shared memory system called SCASH[HIH⁺00], and a multi-threaded programming language called MPC++[IHK⁺93, IHK⁺94, IHS⁺95, IHT⁺96, TISY].

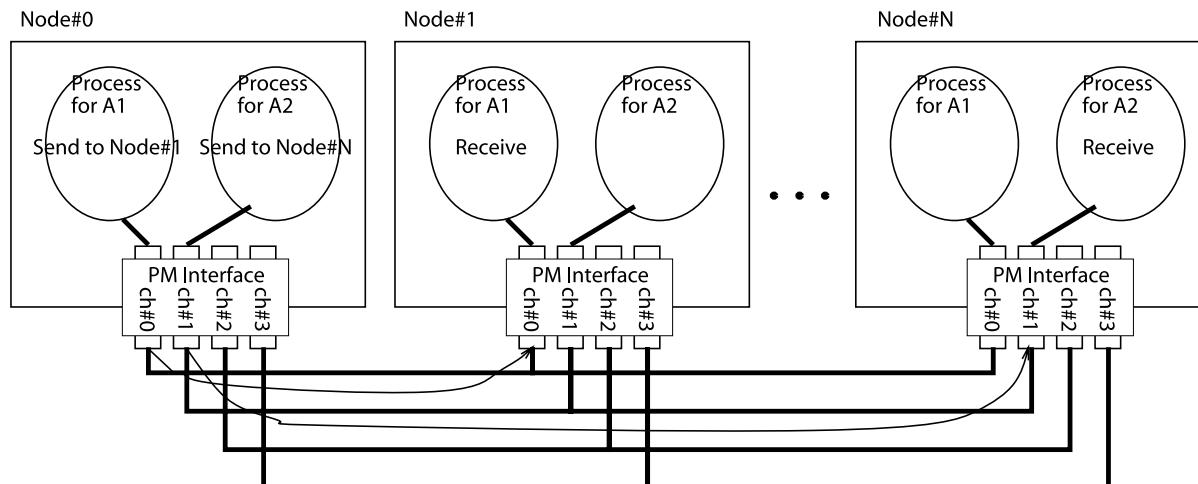


Figure 3.2: Virtual Networks

3.2.1 A Low Level Communication Facility: PM

The PM communication facility[THI96, THIS97, TOHI98] is realized using a user-level communication. To achieve low latency and high bandwidth communication, PM directly accesses the Myrinet network interface to eliminate kernel traps and data copies between kernel and user spaces[THIS97]. PM consists of a user-level library, a kernel-level driver, and a communication protocol handler on the Myrinet network hardware. The PM kernel-level driver initializes the Myrinet network interface and maps the SRAM of the interface to the user address space so that the PM user-level library accesses to it directly.

PM provides a virtual network mechanism called PM channel as shown in Figure 3.2 in order to realize a multi-user environment on the SCore software. The channel provides reliable datagram communication, instead of connection oriented communication such as that of TCP/IP. Figure 3.2 shows an example of the virtual network usage. It is assumed that two parallel applications A1 and A2 run on Node#1 through Node#N. The processes for A1 use channel 1 while the processes for A2 use channel 2 in this figure. Each process for the same parallel application uses a PM channel exclusively, and communicates using the same number of PM channels. A communication to another destination node is done using the node number. PM context is also introduced, which stores the status of the PM channel, and enables context switching on the SCore-D gang scheduler.

PM realizes not only message passing but also a remote memory transfer (*Zero-Copy* communication), to provide high bandwidth communication. Table 3.1 shows the original PM application programmable interface.

In remote memory transfer, a message is transferred using the DMA facility of the Myrinet network without any memory copy operation by the host processor. Since the DMA facility accesses the physical memory address space, the user virtual memory must be pinned down to a physical memory location before the message is transferred. If each message transfer involves pin-down and release of kernel primitives, the message transfer bandwidth will decrease since those primitives are quite expensive. The *pin-down cache* technique [TOHI98] which reuses the pinned-down area to decrease the number of calls to pinned-down and release primitives has been proposed and implemented with a Zero-Copy message transfer. The PM kernel-level driver implements the pinned-down and release primitives since the `mlock` and `munlock` primitives are only available under the super user mode.

Table 3.1: Original APIs on PM

Functions	Description
PM Initializing _pmInit()	Initializing PM library
PM Context Processing _pmOpenContext() _pmCloseContext()	Opening a PM Context Closing a PM Context
PM Message Sending and Receiving _pmGetSendBuf() _pmSend() _pmReceive() _pmReleaseReceiveBuf()	Getting a send buffer Sending a message Receiving a message Releasing a receive buffer
PM Remote Memory Processing _pmMLock() _pmMUnlock() _pmVWrite() _pmWriteDone()	Processing Pin-down of user memory Releasing Pinned-down user memory Processing remote memory write operation Checking remote memory write completion

3.2.2 A Multi-Threaded Language: MPC++

MPC++[IHK⁺94, IHS⁺95, IHT⁺96, TISY] is a multi-threaded programming language. MPC++ Version 2 is designed using two levels, level 0 and level 1. In MPC++ Level 0, called the Multi-Thread Template Library (MTTL), parallel description primitives realized by the C++ template feature, without any language extensions, are specified. This provides remote function invocation, synchronization structure, and remote memory access facilities[Ish96].

In MPC++ Level 1, the MPC++ meta-level architecture which enables library designers to provide an optimizer specific to their class/template library in the library header file[TISY], is specified. The library user may use such a high performance library by including it in the header file.

3.2.3 The MPICH-PM Communication Library

MPICH-PM[OHT⁺97] is an MPI library developed for PM. When a message passing library such as MPI is implemented on top of a lower level communication facility that supports the *Zero-Copy* message transfer primitive[OTH98], the message passing library must handle the pinned-down memory area which is a restricted quantity resource under a paging memory system. Deadlock can be caused by allocation of pinned-down memory by multiple simultaneous requests for sending and receiving without some form of control. MPICH-PM has overcome this issue and achieves good performance.

MPICH-PM/CLUMP[TOT⁺99], the successor of MPICH-PM, supports a cluster of multiprocessors, called CLUMP. Using MPICH-PM/CLUMP, The MPI legacy programs run on CLUMP without any modifications. For example, suppose we have a CLUMP consisting of 16 nodes, each of which contains dual processors. MPICH-PM/CLUMP provides an MPI application with 32 processors or 32 processes. The communication between two processes on different nodes is realized by the PM communication facility using a Myrinet network. Message transfer between two processes on one node is handled by the *Direct Memory Copy*[TOT⁺99] technique which copies between the two processes directly using the PM kernel-level driver.

3.2.4 SCore-D

The SCore-D global operating system[HIK+95, HYI+95, HIN+95, HTI+96, HTI98] is implemented as a set of daemon processes on top of a Unix operating system, written in MPC++ without any kernel modification[HTI98]. To effectively utilize processor resources and to realize an interactive programming environment, parallel processes are multiplexed in the processors' space and time domains simultaneously under SCore-D. Parallel processes are gang-scheduled when multiplexed in the time domain. It has been proven that the SCore-D gang scheduler overhead is less than 4 % of the total application execution time[HTI98].

To realize gang scheduling under a communication layer which accesses the network hardware directly, the network hardware status and messages inflight on the network must be saved and restored when switching to another parallel process. This mechanism is called network preemption. By co-designing PM and SCore-D, the network preemption technique has been developed.

3.2.5 SCASH

SCASH[HIH+00] is a software distributed shared memory system using PM library, and memory management functions, such as memory protection, supported by an operating system kernel. It is implemented as a user level runtime library. SCASH is a page-based distributed shared memory system where the consistency of shared memory is maintained on a per-page basis. SCASH is based on the Release Consistency (RC) memory model with the multiple writer protocol.

3.3 The Cluster System Hardware Platform

This section describes the cluster research platforms, consisting of RWC PC Cluster II, RWC SCore Cluster I and RWC SCore Cluster II. The SCore Cluster system software has been developed on these RWC clusters. A four node mini-cluster which has Dual Pentium II 400MHz processors is also used to develop the communication facilities.



Figure 3.3: RWC PC Cluster II

Table 3.2: RWC PC Cluster II Specifications

Number of Processors	128
Processor	Pentium Pro 200MHz
Memory [MB]	256
I/O Bus	PCI
Local Disk	4GB
Networks	Myrinet, Fast Ethernet

3.3.1 RWC PC Cluster II

RWC PC Cluster II[ITH⁺99] was designed to make a cluster system compact and easily maintainable. We use the PCI-ISA passive backplane standard specified by the PICMG (PCI Industrial Computer Manufacturers Group)[PIC].

Figure 3.3 shows a photograph of RWC PC Cluster II, and Table 3.2 shows the specifications of the cluster. All nodes are connected by a Myrinet and Fast Ethernet network.

In this thesis, RWC PC Cluster II is used as a platform to evaluate the scalability of a



Figure 3.4: RWC SCore Cluster I

Table 3.3: RWC SCore Cluster I Specifications

Number of Processors	48
Processor	16 node Dual Pentium III 500MHz, 16 node Alpha 21264 500MHz
Memory [MB]	512
I/O Bus	PCI
Local Disk	9GB
Networks	Myrinet, Gigabit Ethernet, Fast Ethernet

communication facility on Fast Ethernet.

3.3.2 RWC SCore Cluster I

RWC SCore Cluster I was developed to evaluate a heterogeneous environment (different CPU architectures and different types of networks).

Figure 3.4 shows a photograph of RWC SCore Cluster I, and Table 3.3 shows its specifications. RWC SCore Cluster I consists of a 16 node Dual Pentium III 500MHz system



Figure 3.5: RWC SCore Cluster II

and a 16 node Alpha 21264 500MHz system connected by Myrinet, Gigabit Ethernet and Fast Ethernet networks.

In this study, RWC SCore Cluster I is used as a platform to evaluate the scalability of a communication facility using hardware independent techniques on Gigabit Ethernet, and to compare application performance on different networks.

3.3.3 RWC SCore Cluster II

RWC SCore Cluster II was developed to evaluate different types of networks, such as Myrinet and Gigabit Ethernet with a 64 bit PCI bus.

Table 3.4: RWC SCore Cluster II Specifications

Number of Processors	128
Processor	64 node Dual Pentium III 800MHz
Memory [MB]	512
I/O Bus	PCI
Local Disk	9GB
Networks	Myrinet, Gigabit Ethernet, 3x Fast Ethernet

Figure 3.5 shows a photograph of RWC SCore Cluster II, and Table 3.4 shows the specifications of the cluster. RWC SCore Cluster II consists of 64 node Dual Pentium III 800MHz systems with 64 bit 33 MHz PCI buses. All nodes are connected by Myrinet and Fast Ethernet. The first half of the cluster is also connected by Gigabit Ethernet. The latter half is connected by three NICs for Fast Ethernet networks.

In this study, RWC SCore Cluster II is used as a platform to evaluate the scalability of a communication facility that does not depend on hardware, using existing Gigabit Ethernet NICs and multiple NICs.

3.3.4 Application Performance on RWC Clusters

Figure 3.6 shows the Prowat benchmark results comparing RWC clusters with commercial parallel computers used in studies of molecular dynamics. The Prowat benchmark is a part of Amber[AMB], version 5, which was developed by UCSF, and sold as commercial molecular dynamics software.

The execution time of the Prowat benchmark program is represented as the sum of the non-parallel execution time for a data setup and the parallel execution time. The nonsetup time shown in Figure 3.6 represents the time without the setup time included.

Figure 3.6 shows that the SCore Cluster System software runs at a speed comparable to the commercial speed, and, the RWC PC clusters achieve comparable performance to that of commercial parallel computers, such as the Cray T3E, the SGI Origin 2000, and the Hitachi SR2201.

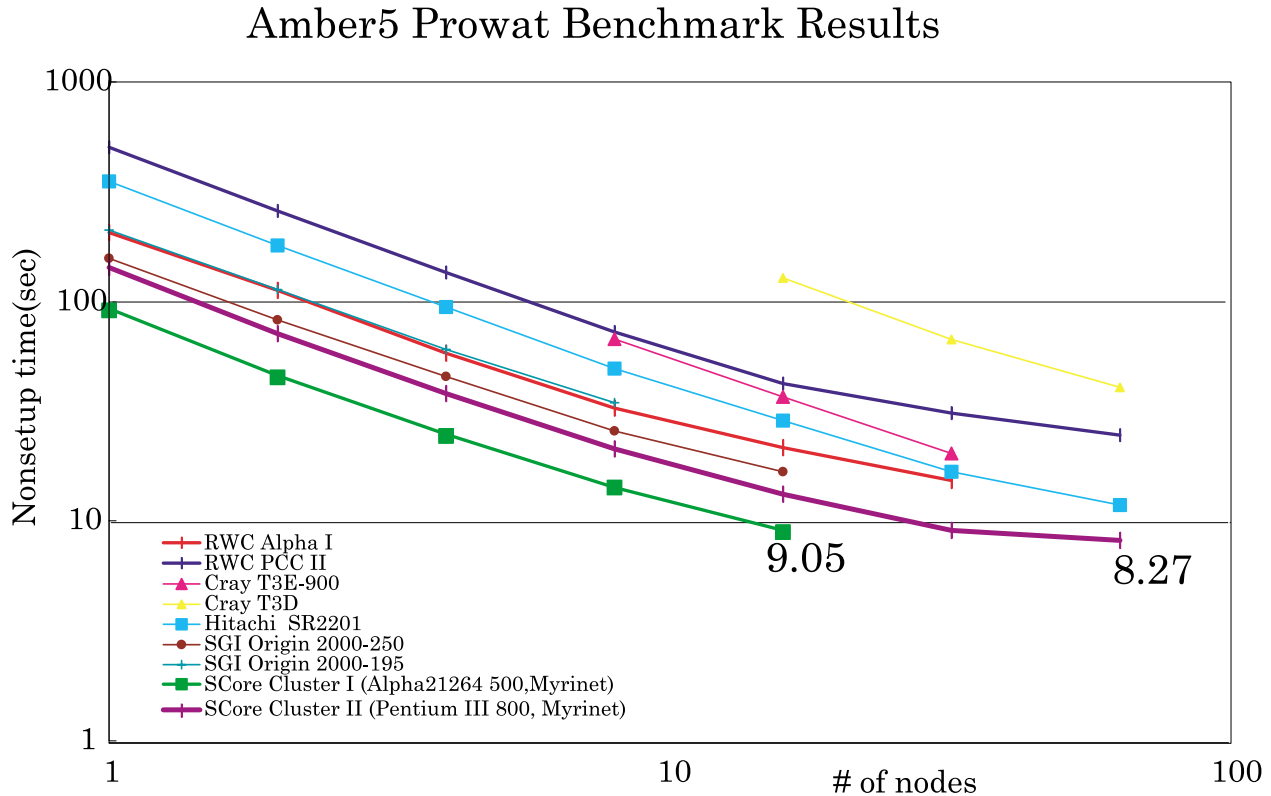


Figure 3.6: Amber Version 5 Prowat Benchmark Results

3.4 The SCore Software on Commodity Networks

The SCore software depended on Myrinet network, and did not support network heterogeneity. But, supporting other networks, especially local area networks, was required in order to build a seamless system.

The appearance of Gigabit Ethernet has made Gigabit class communication possible, so the RWC project has started to develop the SCore software on Gigabit Ethernet to support network heterogeneity.

The goals of the communication facility on Gigabit Ethernet are as follows:

- Realizing high communication performance.
- Supporting the coexistence of a cluster communication protocol with other existing protocols such as TCP/IP.

3.5 Summary of This Chapter

- This thesis has done as a part of the research topics of the SCore Cluster System software.
- RWC clusters have achieved performance higher than that of a commercial parallel computer.

Chapter 4

Hardware Dependent Designs

This chapter proposes a method to minimize the information exchange cost between the host and network interface card (NIC). A reliable communication protocol must be implemented on a communication facility because Gigabit Ethernet does not guarantee message transfer at the hardware level. There are significant design choices: where should a reliable protocol be processed, in the host CPU or in the NIC CPU; and where should the data structures for managing the send and receive buffers and triggering be allocated, in the host or in the NIC memory. Since those structures are shared resources, the cost of host and NIC information exchange is crucial in performance.

In this approach, the cost of TCP/IP processing is analyzed using a model of the information exchange of TCP/IP, and the following method and protocol have been developed:

- A design method for minimizing such information exchange cost, There are significant design choices: where should a reliable protocol be processed, in the host CPU or in the NIC CPU; and where should the data structures for managing of the send and receive buffers and triggering be allocated, in the host or in the NIC memory. Since those structures are shared resources, the cost between host and NIC information exchange is crucial in performance.
- A reliable light-weight communication protocol called “GoBack-N for PM”

Using these method and protocol, the GigaE PM communication facility has been designed, and its performance evaluated.

Table 4.1: Data Transfer Cost Between the Host and NIC Memories and Interrupt / System Call Processing

Processing	Costs (μs)
N words (4bytes/word) transfer:	
from Host to Host by Host Processor	$0.03 \times N$
from Host to NIC by Host Processor	$0.25 \times N$
from NIC to Host by Host Processor	$0.57 \times N$
from Host to NIC by NIC DMA	$2.39 + 0.03 \times N$
from NIC to Host by NIC DMA	$2.00 + 0.03 \times N$
Interrupt Processing	6.5
System Call Processing(ioctl)	1.0

4.1 Network Protocols and NICs

In network communication using existing protocols such as TCP/IP, when a message arrives at a host, the message is passed to the host memory via an I/O bus, e.g., PCI. The host processor handles the upper protocol layers, such as TCP/IP. Such protocol handling layers across the NIC and the host incur some processing costs.

A modern NIC has an on-board processor and memory so that the data-link layer can be handled by the NIC. Table 4.1 shows the basic costs of data transfer between host and NIC memories and interrupt/system call processing. These costs were measured by executing test programs on the NIC processor and the device driver for the host, using PCI Gigabit Ethernet NIC plugged into a Pentium II 400 MHz PC (Red Hat 5.1, Linux 2.1.131). The Gigabit Ethernet NIC is manufactured by an Essential Communications Inc.

In this section, the information exchange cost of TCP/IP is estimated using a model in order to provide information to be used to design a communication facility.

4.1.1 Cost Estimation

Figure 4.1 and the following description show the data and control flow for handling TCP/IP in a typical implementation. This flow does not reflect the actual implementation, but rather focuses on the information movement between the host and the NIC. It is assumed that the message send and receive buffers, and their descriptors are located in

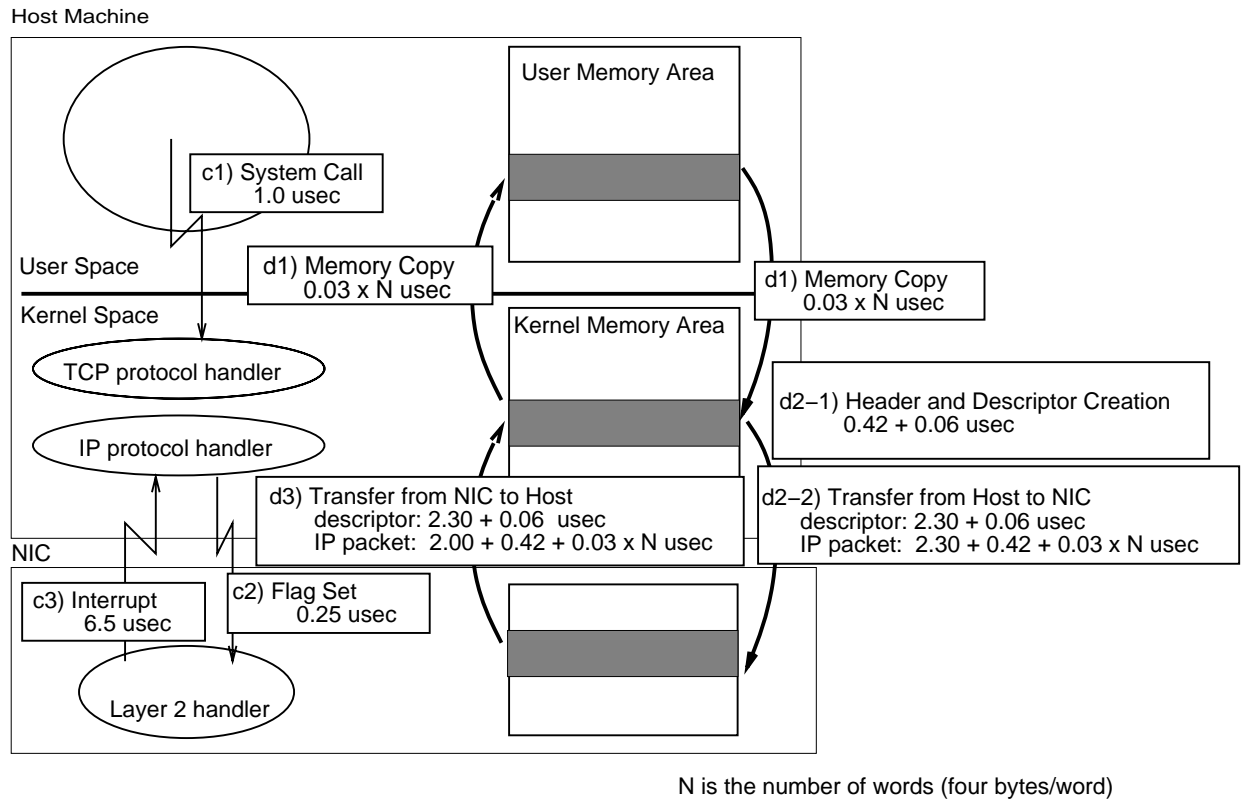


Figure 4.1: Data and Control Flow in TCP/IP Handling

the host memory. The cost is calculated based on Table 4.1.

- **Transfer between the user and kernel spaces.**

The user data is transferred to the kernel space by invoking a system call. It requires a system call and memory copy cost as shown in c1) and d1) of Figure 4.1. The total cost (C_a) is as follows.

$$C_a = 1.0 + 0.03 \times N(\text{words}) \mu s.$$

- **An IP packet transfer from the host to the NIC.**

TCP/IP and an Ethernet headers are assembled with the data, and then the IP packet is transferred to the NIC. A pointer, pointing to the header and data areas, is stored in a message sender descriptor. In d2-1) of the figure, it is assumed that the header creation cost is the same as the header copy cost. The host tells the NIC

that the new descriptor is available. This cost is shown in c2). Then the NIC issues a DMA so that the packet header and data are transferred to the NIC, whose cost is shown in d2-2). The total cost (C_b) is as follows.

$$C_b = 5.81 + 0.03 \times N \mu s.$$

- **An IP packet transfer from the NIC to the host.**

When an IP packet arrives at the NIC, the packet is transferred to the host. Then the data-link layer handler in the NIC informs the host of the packet arrival by issuing an interrupt signal. Those two costs (C_c) shown in d3) and c3) are as follows.

$$C_c = 11.28 + 0.03 \times N \mu s.$$

4.1.2 Discussion

According to the above cost estimation, the cost for an N word message transfer from the sender to the receiver is given by the following expression.

$$2 \times C_a + C_b + C_c = 19.09 + 0.12 \times N \mu s.$$

This shows that a one word message transfer requires $19.21 \mu s$. For the total cost of a message passing from the sender to the receiver, the TCP/IP protocol handling cost in the host processor and the data-link layer protocol handling in the NIC must be added. To implement the TCP/IP protocol, control packets, such as ACK, are passed between the sender's and receiver's protocol handlers, where the communication cost between the host and the NIC is $17.09 \mu s$ according to the cost estimation.

To ensure a high performance communication facility, the cost between the host and the NIC shown above must be reduced as much as possible.

4.2 Design Objectives of GigaE PM

The GigaE PM design objectives are summarized as follows:

- **Providing a simple network protocol for a parallel application:**

A network protocol must not require large message buffers in order to communicate with many processes.

- **Guaranteeing the reliability and FIFO-ness of message arrival:**

Since Gigabit Ethernet does not guarantee message arrival, the network protocol must support reliable communication.

- **Realizing low information exchange overhead between the NIC and the host:**

As previously stated, the location of the information shared by the host and the NIC dominates the information exchange overhead.

- **Ensuring the coexistence of the GigaE PM protocol with other protocols such as TCP/IP:**

Since the communication facility is used in the LAN environment where other protocols such as TCP/IP are also used, the facility must support both the dedicated protocol for high performance computation and traditional network protocols.

4.3 GigaE PM Design

4.3.1 Virtual Networks and API

The GigaE PM communication facility is designed to adapt to a parallel application. It provides a channel to support a *virtual network* which has been introduced in PM[THIS97] as described in section 3.2.1. A communication on the channel is same as datagram communication with reliable reception, not connection oriented communication, such as TCP/IP. Each process of a parallel application uses the same channel number exclusively which represents a virtual network. The number of channels depends on the NIC's hardware resources. In the current implementation, four channels are supported. To accommodate the fact that parallel processes equal to more than the number of channels may run, the SCore global operating system that multiplexes PM[THIS97] channels was developed.

The API of the GigaE PM is based on PM[THIS97]. Therefore, programs written for the SCore Cluster System Software, including MPI, can be executed on top of the GigaE PM. It should be noted that the GigaE PM is mainly used to implement upper level communication libraries such as MPI.

4.3.2 Where Should a Communication Protocol be Processed?

There are two choices as to where a communication protocol is processed: the host CPU or the NIC CPU.

Handled in the host CPU: Because the host CPU is generally faster than NIC CPUs, protocol processing on the host CPU will be faster than that on the NIC CPU. However control messages to maintain reliable communication protocols must be transferred through an I/O bus, such as the PCI bus, between host memory and NIC memory. This transfer cost is more than $2 \mu s$ on the Essential Communications PCI Gigabit Ethernet NIC as shown in Table 4.1.

Handled in the NIC CPU: If a NIC has a programmable on-board CPU and memory, a communication protocol can be implemented on the NIC. This choice can eliminate the overhead of control message transfer through an I/O bus such as a PCI bus.

A method by which the communication protocol should be handled on the NIC CPU is selected in order to minimize the data exchange cost between the host and the NIC.

4.3.3 Reliable Communication

To guarantee message delivery and FIFO-ness, a protocol called *Go-Back N for PM* using **GO back N** with **STOP and GO** flow control has been adopted.

In the **GO back N** protocol, the sender may send the i th to the $i + N$ th data messages to the receiver without waiting for an acknowledgment of message reception from the receiver. That is, the $i - 1$ th ACK message from the receiver has been received by the sender. The receiver sends the i th ACK message back when the i th data message has been received by the receiver. When the sender receives the i th ACK message, the sender may send the $i + 1$ th to $i + 1 + N$ th data messages to the receiver.

If the sender does not receive the i th ACK message after a certain time, the sender sends the i th and the following data messages again. This happens when the i th data or ACK message has lost. The sender needs to have a buffer area to keep the i th to $i + N$ th messages, but the receiver does not need to have a buffer area to keep the N data messages.

When the receiver receives a data message whose sequence number is larger than i , the received data is discarded, and then a **LOSE** message is sent back to the sender. This differs from the TCP/IP sliding window protocol.

When the receiver's buffer area becomes full, a **STOP** message is sent to stop the sender. The receiver will send a **GO** message when the receiver has enough buffer area available. This protocol is well known as **STOP and GO** flow control.

The network protocol will be described in detail in Appendix B.

4.3.4 Information Exchange between the Host and the NIC

As described in section 4.1, information exchange between the host and the NIC is crucial in the design of a high bandwidth and low latency communication facility. The information is exchanged through a message descriptor. This entry has a message address, a message size, and a sender or receiver identifier depending on the type of message.

First of all, the location of descriptors and their access methods are designed as follows:

- **Descriptor Location**

The message descriptors can be located on either the host or the NIC. Since the descriptors are accessed by both the host and the NIC, they should be located where the total access time cost is minimized.

There are two choices:

- 1 The descriptors are in the host memory.
- 2 The descriptors are in the NIC memory.

When the descriptors are in the host memory, NIC DMA is required. When in the NIC memory, the descriptors need to be written by the HOST CPU. From the costs as shown in Table 4.1, the access cost is calculated (Table 4.2). From the results as shown in Table 4.2, the descriptors are located on the NIC.

- **Descriptor Access**

If the user process writes a send message descriptor and informs the NIC that a new

Table 4.2: Access Cost by Descriptors Location

Location (Method)	Costs (4 Words Transfer)
1. Host Memory (Using NIC DMA)	2.4 μs
2. NIC Memory (Using HOST CPU)	1.3 μs

descriptor is ready, no kernel calls are required in sending a message. Two security issues must be considered in this access:

- 1 send message descriptors must be isolated from other user processes so that the user process only accesses the proper area,
and
- 2 the send message descriptor must be verified because the user process might have specified a bad message address.

In the Essential Communications PCI Gigabit Ethernet NIC, the host can access to the entire memory in the NIC by simply writing to a control register of the NIC. If all send message descriptors are located on the NIC, accessing to a send message descriptor requires writing to the control register. If the user has a write-access right to the register, the user may write to other registers. Thus, the security issue on user access as described above cannot be resolved on the Essential Communications NIC. Therefore, GigaE PM provides a kernel function that accesses a message descriptor on the NIC. Since the host can access a restricted memory area without writing to a control register on the Essential Communications NIC, the receive message descriptors are stored in this area so that the user process may read the message descriptor without a kernel call.

- **Trigger**

The following methods for informing the NIC/host by the host/NIC must be considered:

From the Host to the NIC :

Two methods can be used:

- 1 The host writes a flag in the memory of the NIC.
- 2 The NIC polls a flag area in the host.

Table 4.3: Access Cost by a Flag Location

Location (Method)	Costs (1 Word Transfer)
1. NIC Memory (Using HOST CPU)	0.25 μs
2. Host Memory (Using NIC DMA)	2.41 μs

From the costs as shown in Table 4.1, the access cost is calculated (Table 4.3). According to Table 4.3, we decided to adopt the method where the host processor writes a flag in the memory of the NIC.

From the NIC to the Host :

Here, there are three methods:

- 1 The NIC writes a flag in the host's memory using the NIC DMA.
- 2 The host polls a flag area on the NIC.
- 3 The NIC issues an interrupt signal to the host.

Table 4.4: Trigger Cost From the NIC to the Host

Method	Costs
1. Using a Flag in the HOST Memory	2.03 μs
2. Using a Flag in the NIC Memory	0.57 μs
3. Using Hardware Interrupt	7.07 μs

From the costs as shown in Table 4.1, the access cost is calculated (Table 4.4). According to Table 4.4, the host polls a flag area on the NIC.

4.3.5 Host CPU Polling effects on PCI DMA transfer

Because frequent polling executed by the host CPU may cause degradation to the PCI DMA transfer performance, the degradation of DMA transfer performance caused by polling is measured.

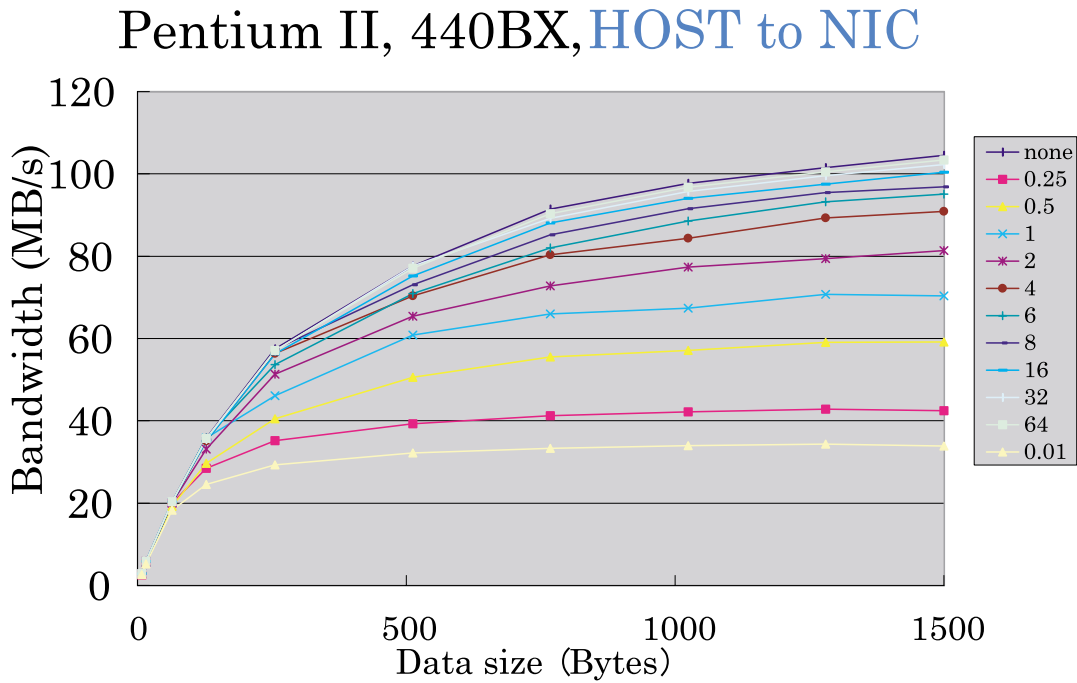


Figure 4.2: Host CPU Polling effects on PCI DMA transfer: Host to NIC

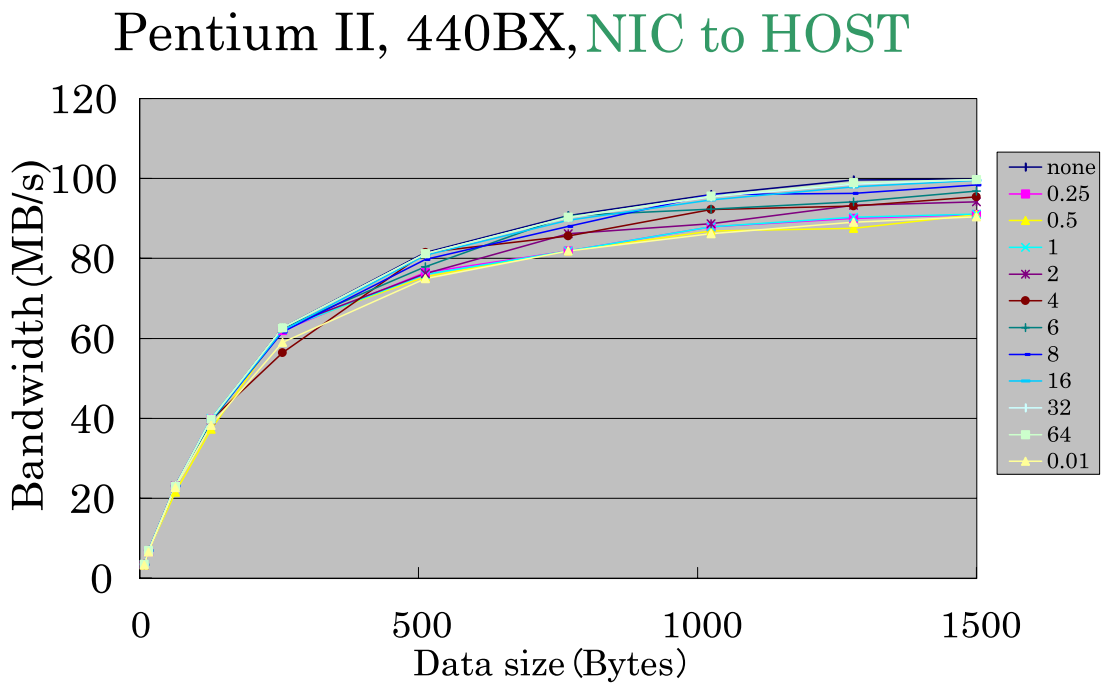


Figure 4.3: Host CPU Polling effects on PCI DMA transfer: NIC to Host

Figures 4.2 and 4.3 show the results with the Intel 440BX chipset on the environment shown in Table 4.7. The legend shows the period of polling in μs .

From Figures 4.2 and 4.3, when the polling period is set to $0.25 \mu s$, the PCI DMA bandwidth from host to NIC decreased to around 40 MB/s. The period should be set to over $4 \mu s$ to realize over 80 MB/s PCI DMA bandwidth with the Intel 440BX chipset.

4.4 GigaE PM Implementation

4.4.1 Overview of the GigaE PM Implementation

This subsection describes an overview of GigaE PM implementation. GigaE PM consists of GigaE PM firmware on a Gigabit Ethernet NIC, a kernel driver for GigaE PM and the GigaE PM user level library. An overview of GigaE PM is described as follows:

- The application interface is compatible with PM [THIS97, TOHI98] on Myrinet. The basic application-program-interface is as follows:
 - `_PM_getsendBuf()`: gets a send message buffer
 - `_PM_sendmsg()`: sends a message and releases the buffer
 - `_PM_receive()`: receives a message
 - `_PM_putreceiveBuf()`: releases the receive message buffer
- Send and receive message buffers are pre-allocated in the host memory, always pinned-down to physical memory, and mapped into user space using `mmap()`.
- Send and receive descriptors are in the NIC memory. The application writes a buffer address, a size and a destination to a descriptor using `ioctl()`.
- The NIC registers are mapped to read-only user-memory-space.
- The NIC firmware multiplexes (or de multiplexes) Ethernet frames for cluster and other communications. The cluster communication uses a special Ethernet-type frame.
- The maximum transfer units (MTU) of GigaE PM is 1,468 bytes, and currently, GigaE PM does not support fragmented packets.

- The polling period is set to $4 \mu s$.

4.4.2 Overview of GigaE PM Message Handling

This subsection describes an overview of GigaE PM message handling (Figure 4.4).

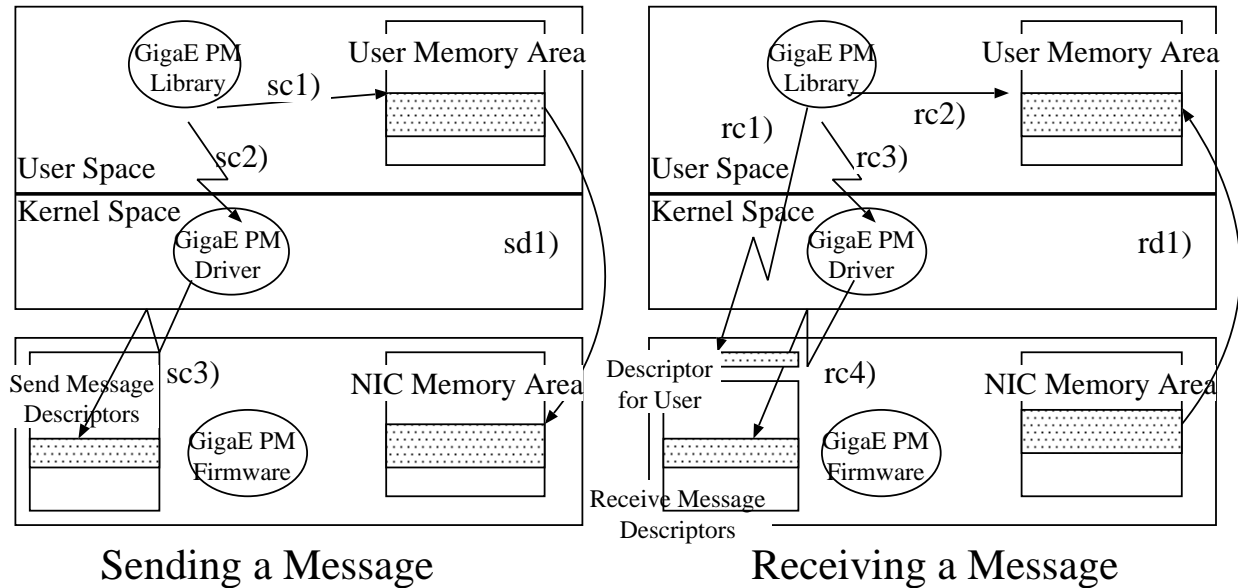


Figure 4.4: Descriptors, Host, and NIC in GigaE PM

Sender:

- 1 The user program gets a send message buffer using `_PM_getsendBuf()` as shown in sc1) of Figure 4.4, and writes data into the buffer. Then the user sends the message using `_PM_sendmsg()`.
- 2 The `_PM_sendmsg()` function, realized as the GigaE PM library, invokes the GigaE PM driver in sc2) so that the driver writes message data (message address, message length and receiver identifier) to a descriptor on the NIC as shown in sc3).
- 3 The NIC will transfer the message in the user space to a NIC memory area as shown in sd1).

Receiver:

- 1 The `_PM_receive()` function, in the GigaE PM library, polls the arrival flag in the NIC without any kernel trapping as shown in rc1) of Figure 4.4. If the arrival flag has been set the receive message descriptor is read so that the message buffer address, message size, and the sender identifier are obtained as shown in rc2) of Figure 4.4.
- 2 When the NIC receives a new incoming message, the message is transferred to a user message buffer whose area has been registered by the initialization routine of the GigaE PM library in rd1). The NIC updates the arrival flag.
- 3 The `_PM_putreceiveBuf()` function, realized by the GigaE PM library, invokes the GigaE PM driver in rc3) to inform the NIC that the message buffer has been released in rc4).

4.4.3 TCP/IP and Other Protocol Support

The GigaE PM network protocol is handled on the NIC, while other network protocols such as IP are handled on the host, the same as other Ethernet NICs do. When a network packet whose type is not GigaE PM arrives, the packet is transferred to the host and a handler on the host is triggered. Send and receive descriptors for TCP/IP and other protocols exist in NIC memory so that the GigaE PM network protocol can be processed simultaneously with TCP/IP.

4.4.4 Comparison of Cost and Buffer Usage

Table 4.5 shows the comparison of cost in GigaE PM with that in TCP/IP for processing an N word message. The costs in the GigaE PM are estimated from subsection 4.4.2.

In Table 4.5, the one word message transfer cost from the sender to the receiver in GigaE PM is $9.89 \mu s$. The cost in TCP/IP is $19.21 \mu s$. The cost reduction is 48.5% in GigaE PM. In TCP/IP, the interrupt processing cost ($6.5 \mu s$, Figure 4.1 c3) and descriptor transfer cost using NIC DMA ($4.72 \mu s$, Figure 4.1 d2-2, d3) occupy 58.4% of the total cost. This is because GigaE PM uses polling method instead of hardware interrupt used in TCP/IP, and the descriptor location of GigaE PM is in the NIC memory so that descriptor transfer is not needed by NIC DMA.

Table 4.5: Comparison of Cost in an N Word Message

Description	TCP/IP (μs)	GigaE PM (μs)
System Call and Copy	$1.0 + 0.03 \times N$	1.0
Sending a Message	$5.81 + 0.03 \times N$	$3.30 + 0.03 \times N$
Receiving a Message	$11.28 + 0.03 \times N$	$4.53 + 0.03 \times N$
Sending and Receiving a Message	$19.09 + 0.12 \times N$	$9.83 + 0.06 \times N$
Sending and Receiving an ACK	17.09	0

Table 4.6: Comparison of Memory Usage in an N node Cluster

Description	TCP/IP (KB)	GigaE PM (KB)
N Node Cluster		
Send Buffer Size	$1.5 \times M \times N$	120
Receive Buffer Size	$1.5 \times M \times N$	128
Control Structure Size	$0.274 \times N$	$0.023 \times (N + 1)$
128 Node Cluster(M=8)		
Send Buffer Size	1536	120
Receive Buffer Size	1536	128
Control Structure Size	35.072	2.967

Table 4.6 shows the comparison of memory usage in GigaE PM and memory usage in TCP/IP in an N node cluster. In Table 4.6, the MTU of TCP/IP is 1.5KB, and $1.5 \times M$ shows the window size of TCP/IP.

In Table 4.6, the send and receive buffer memory size of GigaE PM does not depend on the number of nodes. GigaE PM requires send and receive buffer memory whose size needs only 8.01 % of that of TCP/IP, and control structure memory whose size needs only 8.46 % of that of TCP/IP in a 128 node cluster.

4.5 GigaE PM Evaluation

The basic performance, bandwidth and latency, and MPI application performance using NAS parallel benchmarks are measured and compared with those of TCP/IP in this section.

4.5.1 Evaluation Environment

Table 4.7 shows the evaluation environment of GigaE PM. The evaluation of GigaE PM is done with or without Ethernet switch. The evaluation of TCP/IP always uses Ethernet switch.

Table 4.7: Evaluation Environment for GigaE PM

Hardware	Pentium 400 MHz, 440BX chipset, 256 MB SDRAM memory
NIC	Essential PCI Gigabit Ethernet NIC model EC-440-SF (32 bit 33 MHz PCI, DMA for PCI, SEEQ 8100 MAC and MEM 1MB)
Switch	Extreme's Summit 1
Host OS	Red Hat 5.1 (Linux 2.1.131)

NAS parallel benchmarks are used as application programs. The Numerical Aerospace Simulation (NAS) Parallel Benchmarks (NPB) are a set of 8 programs designed to evaluate the performance of parallel supercomputers. NPB will be described in detail in Appendix C.

4.5.2 PM Bandwidth

The bandwidth is shown in Figure 4.5. The figure contains performance data for GigaE PM, TCP/IP on GigaE PM, and TCP/IP developed by the Essential Communications company. GigaE PM has achieved a 56.7 Mbytes/sec bandwidth in the case of a 1,468 byte message. In contrast to GigaE PM, TCP/IP on GigaE PM achieves 28.05 Mbytes/sec, and TCP/IP on the Essential achieves 32.96 Mbytes/sec in the case of a 1,280 byte message. The communication bandwidth of GigaE PM is 1.7 times faster than that of TCP/IP.

4.5.3 PM Round Trip Latency

Figure 4.6 shows the round trip time cost. GigaE PM achieved a 48.3 μs round trip latency with a four byte user message. When a Summit 1 switch is inserted between the

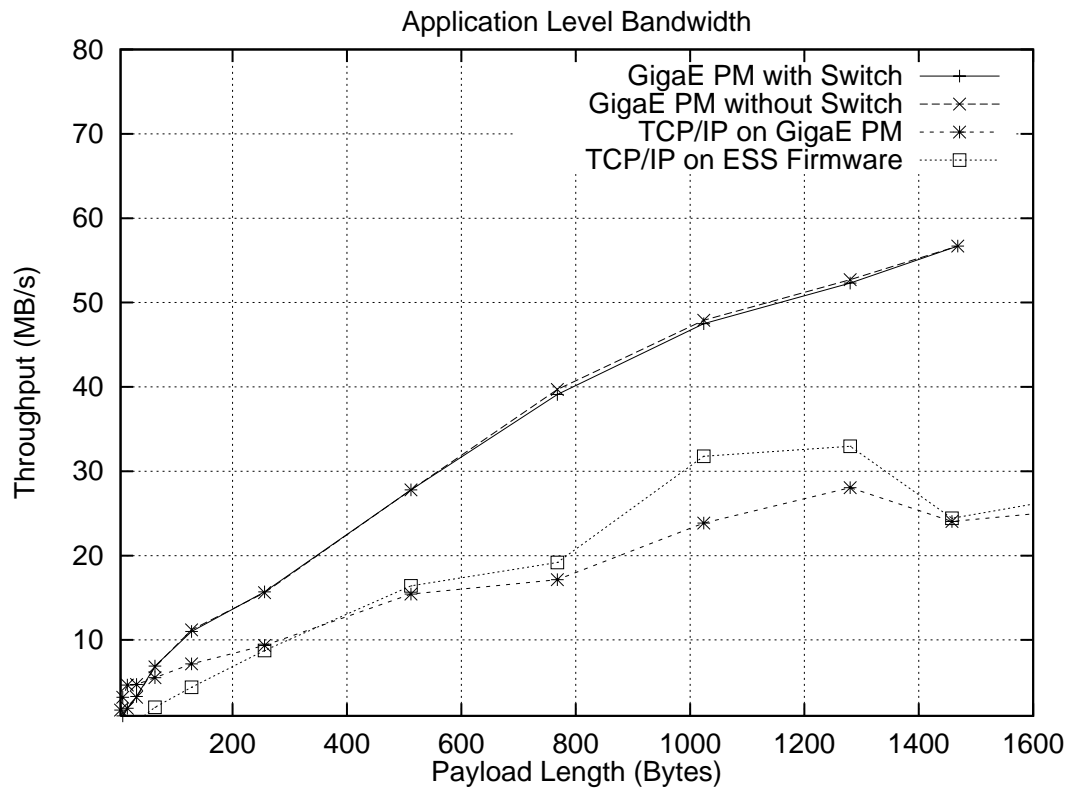


Figure 4.5: GigaE PM Bandwidth

two hosts, the latency increases to $58.3 \mu s$. This means that the message transfer latency on the switch is $5 \mu s$.

The round trip latency of TCP/IP on the Essential firmware is $414.0 \mu s$ while the TCP/IP latency on GigaE PM is $131.4 \mu s$. GigaE PM realizes better performance coexisting with the high performance network protocol. The communication latency of GigaE PM is about one third of that of TCP/IP.

4.5.4 NAS Parallel Benchmarks

MPICH-PM (based on MPICH 1.0.11) on top of GigaE PM was implemented, and the application performance using the CG (Conjugate Gradient) and IS (Integer Sort) programs of the NAS parallel benchmarks, version 2.3 is evaluated. The MPI for TCP/IP was MPICH 1.0.11 with the `ch_p4` device.

Figures 4.7 and 4.8 show the performance of CG and IS whose sizes are class S. MPI on

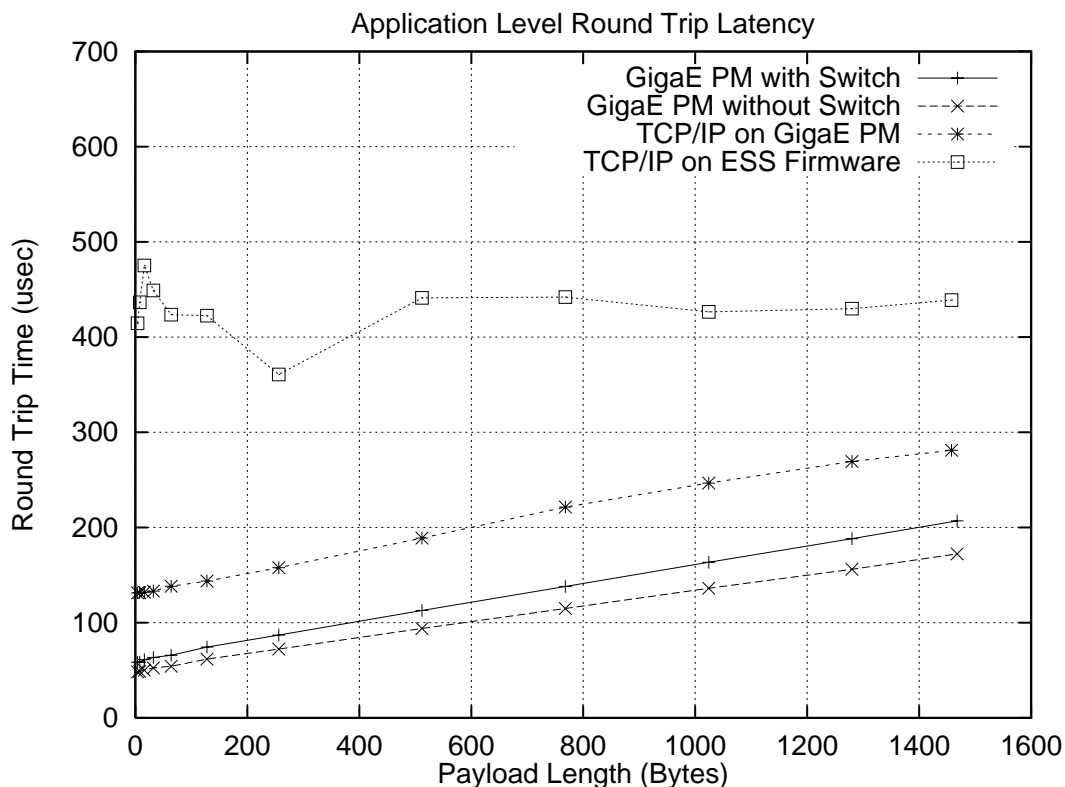


Figure 4.6: GigaE PM Round Trip Time

top of GigaE PM achieves a 3.63 fold speedup for CG with a 2.0 fold speedup for IS on 4 nodes. The MPI on top of TCP/IP achieves a 1.72 fold speedup for CG with a 1.37 fold speedup for IS on 4 nodes. The results show that the IS performance on the GigaE PM is 1.8 times faster than that on TCP/IP. Because the CG and IS benchmarks are latency and bandwidth sensitive, these results reflect the low latency and high bandwidth of GigaE PM.

4.6 Related Work of GigaE PM

Many high performance communication facilities have been developed. AM[T. 92][T. 94], AM-II[CMC97], FM[SCO95], BIP[PT98], VMCC-2[DBC+97] and PM[THIS97] are based on Myrinet. Unlike the Gigabit Ethernet, the Myrinet network supports reliable message transfer at the hardware level. Thus, those facilities do not need to worry about a message being lost in the network.

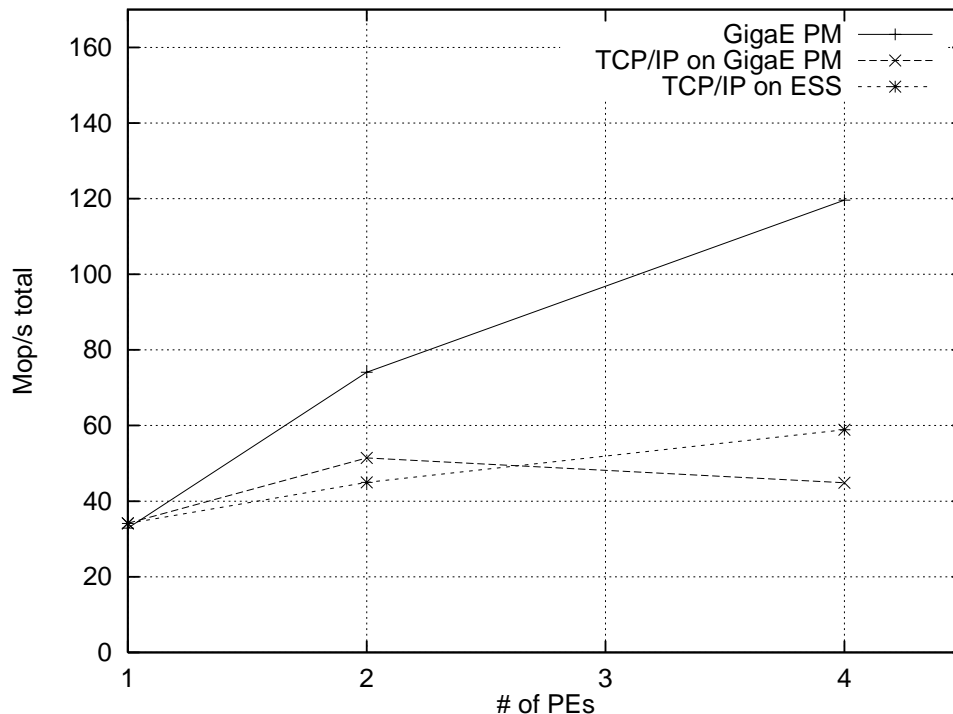


Figure 4.7: CG Class S on GigaE PM

To decrease the kernel trapping overhead, PM and U-Net[BBVvE97] don't use system calls. PM achieves a 15 microsecond round trip time and 113 MBytes/sec bandwidth in a Myricom Myrinet network. Thus, the kernel trapping overhead is crucial. However, in the GigaE PM, the kernel trapping overhead is $2.0 \mu s$ of $48.3 \mu s$ round trip time, or only 4.1 % of round trip time overhead. It is not considered crucial.

VIA[VIA], the Virtual Interface Architecture, is being widely implemented in Gigabit class networks on the Microsoft Windows operating system. VIA is designed so that other communication facilities such as sockets[Ste] and MPI are implemented on top of VIA. VIA supports connection oriented communication. Reliable communication support is an option according to the VIA specification Version 1.0. As described in section 4.1, if a reliable connection oriented communication facility is realized using an unreliable network such as Gigabit Ethernet, larger communication buffers are required. Such a facility cannot be implemented on a NIC. Thus, we think that it is difficult to support high performance communication for parallel applications, especially data parallel applications on connection

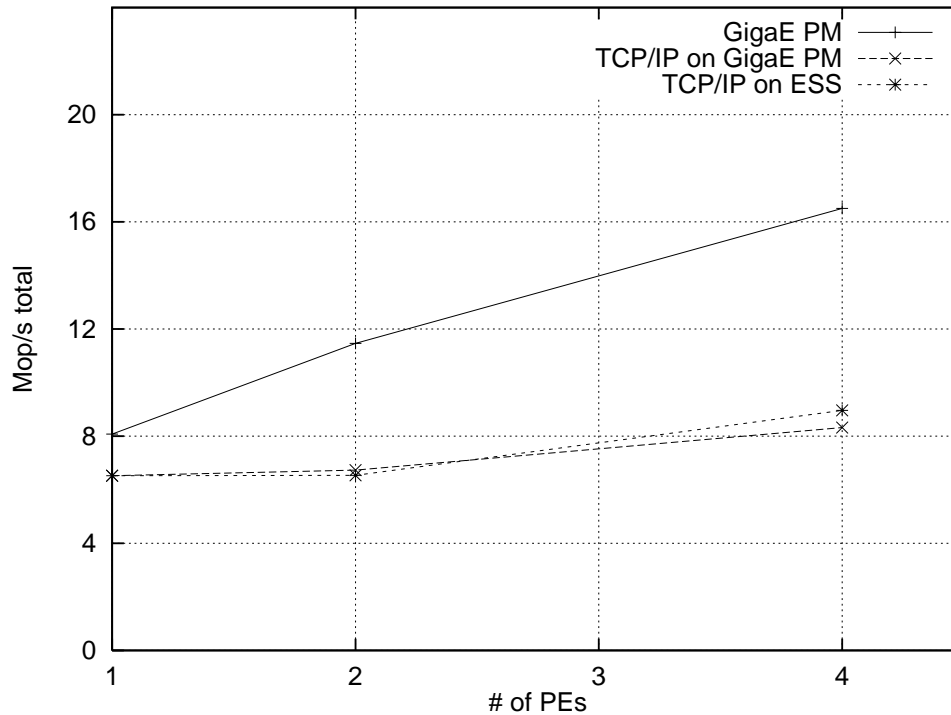


Figure 4.8: IS Class S on GigaE PM

oriented communication.

Since the GigaE PM protocol is simple and does not require large message buffers, unlike other connection oriented communication facilities, the handler is implemented on a NIC and provides reliable communication on top of the Gigabit Ethernet.

4.7 Conclusions of This Chapter

In this chapter, it has been pointed out that information exchange between the NIC and the host is crucial in the design of high performance communication for parallel applications on clusters of computers using Gigabit Ethernet. Therefore, a high performance communication facility called *GigaE PM* has been designed. A prototype system has been implemented using an Essential Communications Gigabit Ethernet card.

The GigaE PM facility provides not only a reliable high bandwidth and low latency communication function but also supports existing network protocols such as TCP/IP. Using

this facility, a high performance cluster system is constructed on a distributed environment where parallel applications coexist with distributed applications.

The performance results show that a 48.3 microsecond round trip time for a four byte user message and a 56.7 MBytes/sec bandwidth for a 1,468 byte message are achieved. The communication latency of GigaE PM is about one third of that of TCP/IP, and the communication bandwidth of GigaE PM is 1.7 times faster than that of TCP/IP.

GigaE PM also supports the TCP/IP protocol without decreasing the performance of the original Gigabit Ethernet card. The performance results, using NAS parallel benchmarks, show that the IS class S performance on the GigaE PM is 1.8 times faster than that on TCP/IP.

GigaE PM is not limited only to the Essential Communications NIC, rather it is a general facility which can be implemented on other modern NICs.

In the results of this study, a communication facility which depends on a NIC hardware should be implemented as follows:

- The NIC should have a programmable on-board CPU and memory in order to implement a communication protocol on the NIC. *Go-Back N for PM*, a light-weight protocol is proposed so that it can be implemented on a NIC CPU.
- The descriptors of the message buffer should be located in the NIC memory in order to minimize descriptor access cost caused by the NIC.
- A polling method should be used in triggering from the NIC to the host. But the short polling period may cause PCI DMA performance degradation.

Chapter 5

Hardware Independent Designs Using Existing NICs

This chapter proposes software designs that do not depend on NIC hardware using existing NICs. A Reliable protocol must be implemented on a host CPU using an existing NIC, so, the cost of the existing communication protocol, TCP/IP, is analyzed. Then, new software techniques are proposed to achieve higher performance than that using an existing protocol such as TCP/IP. These techniques do not require any special hardware or hardware specific device drivers in order to be adapted to many kinds of network interface cards. PM/Ethernet has been implemented using these techniques, and then evaluated.

5.1 Design Objectives of Cluster Communication on a Commodity Network

The characteristics of commodity networks are summarized as follows:

- Many vendors provide many kinds of NIC hardware.
- The lifetime of hardware is relatively short because of vendor competition. So, the same hardware may not be available in the future.

The design objectives of cluster communication on a commodity network can be summarized as follows:

- 1 Applicability to many kinds of NIC hardware.

Currently, if we develop a cluster communication facility which depends on hardware, we must develop and support a number of NICs. Therefore, cluster communication facilities for a commodity network should be independent of hardware in order to apply to many kinds of NICs. This approach uses a device driver without any modification to the source code of the device driver, so the performance depends on the device driver code.

- 2 Coexistence of a cluster communication protocol with other existing protocols such as TCP/IP.

Since cluster communication is used in the LAN environment where other existing protocols such as TCP/IP are also used, both the cluster communication protocol for high performance computation and traditional network protocols should be processed on the same NIC.

5.2 TCP/IP Protocol Processing Overhead

As shown in Figure 5.1, typical protocol handling layers in Linux and Unix-based operating systems contain NICs, device drivers, a data link layer such as Ethernet, IP and TCP protocol handling layers, and a Socket layer. In this section, performance in those protocol layers is measured and then the protocol handling overheads are analyzed. Table 5.1 shows the evaluation environment.

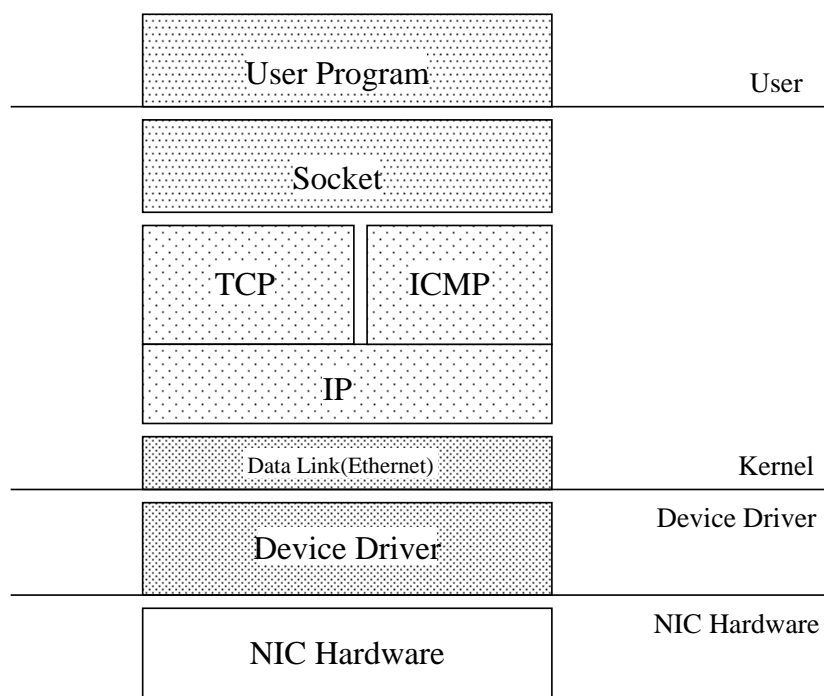


Figure 5.1: TCP/IP Protocol Processing in Unix

Table 5.1: Evaluation Environment for TCP/IP

Hardware	Pentium III 500MHz (440BX chipset, 512MB SDRAM)
NIC	Packet Engines G-NIC II 33 MHz clock, 64 bit PCI using 32 bit PCI slot
Host OS	Redhat 6.1 Linux (2.2.12 kernel)
Device Driver for G-NIC II	hamachi.c:v0.11 8/21/99 Written by Donald Becker

5.2.1 Performance in Protocol Layers

To investigate the communication performance of each protocol layer, round trip time in the data link layer, IP, and TCP/IP protocol layers was measured. Bandwidth in the data link layer and TCP/IP protocol layer was also measured.

The Netperf program[NET] was used to measure the TCP/IP round trip time. The ICMP protocol was used to measure the IP round trip time. To measure the round trip time and

bandwidth in the data link layer of Ethernet, the following programs were implemented in the device driver:

- Round trip time: The sender sends an Ethernet frame to the receiver while the receiver sends back an Ethernet frame whenever the sender's frame is received.
- Bandwidth: The sender sends an Ethernet frame to the receiver while the receiver receives and drops the sender's frame. The bandwidth is measured at the receiver side.

Table 5.2 shows round trip time in the data link, IP, and TCP/IP protocol layers.

Table 5.2: Round Trip Time and Bandwidth in Protocols

Layer	RTT	Bandwidth
Data Link	36.6 μ sec	90.4 MB/sec
IP	58.6 μ sec	-
TCP/IP	89.6 μ sec	46.7 MB/sec

5.2.2 TCP/IP Protocol Processing Overhead Analysis

Table 5.3 shows the result of the TCP/IP protocol overhead which was analyzed as follows:

- System Call and Socket

This overhead was measured by the Pentium III hardware clock counter.

- TCP

Half of the TCP round trip time is the sum of half of the IP round trip time and the TCP protocol handling overhead. Thus the TCP protocol handling overhead is derived as follows:

$$89.6/2 - 58.6/2 = 15.5$$

- IP

Half of the IP round trip time is the sum of the system call and socket overhead,

protocol handler invocation overhead, and half of the data link layer overhead. Thus the IP cost is derived as follows:

$$58.6/2 - 1.6 - 3.2 - 36.6/2 = 6.2$$

- Protocol Handler Invocation

Protocol handler invocation means the protocol context switch from the data link layer to an upper protocol, such as the IP protocol, using a *software interrupt*. This overhead was measured by the hardware clock counter.

- Device Driver

This overhead is given by the execution time of a device driver written in C. It was measured by the hardware clock counter.

- Hardware Interrupt

This cost is given by the time to reach the device driver routine after the NIC asserts the processor interrupt register. This cost was measured using the Essential Gigabit Ethernet NIC firmware. It is 5.9 μ sec because the hardware overhead is 1.6 μ sec, and the Linux interrupt handler accesses the register of the interrupt controller 4 times (1.2 μ). Other software overhead such as context store and device entry search are also figured in. This hardware interrupt cost does not depend on processor clock speed (Appendix A.2).

- NIC+Media

Half of the data link layer round trip time is the sum of the device driver, hardware interrupt, and NIC+media costs. Thus the NIC+media cost is derived as follows:

$$36.6/2 - 4.7 - 5.9 = 7.7$$

According to Table 5.3, the dominant portion of overhead is TCP/IP protocol processing, 48.4% (34.6 + 13.8) of total overhead. The second highest overhead, which the software can reduce, is interrupt processing overhead.

Table 5.3: TCP/IP Overhead

Processing	Overhead	Ratio
System call and socket	1.6 μ sec	3.6 %
TCP	15.5 μ sec	34.6 %
IP	6.2 μ sec	13.8 %
Protocol Handler Invocation	3.2 μ sec	7.1 %
Device Driver	4.7 μ sec	10.5 %
Hardware Interrupt	5.9 μ sec	13.2 %
NIC+Media	7.7 μ sec	17.2 %
Total	44.8 μ sec	100.0 %

5.3 A Protocol Handling Scheme Design

This section proposes a communication protocol handling scheme for a cluster system using a commodity network. An approach realizing a communication facility with machine independence is taken in this scheme in order to adapt to many kinds of NICs, as described in section 5.1.

5.3.1 Protocol Handling for Cluster Computing

In the TCP/IP protocol implementation in Linux and Unix-based systems, the protocol handler is invoked as a kernel thread using a *software interrupt* mechanism.

This is because of the following assumption: a network is slower than other devices such as a hard disk drive. It is a good design assumption on a network computer where a keyboard, a mouse, and hard disk drives are installed.

However, the assumption is not suitable to high performance networks, such as Gigabit Ethernet. Therefore, a method, in which the communication protocol handler is invoked by the network data link layer directly, is proposed in order to minimize message latency.

5.3.2 A Light-weight Reliable Communication Protocol

According to Table 5.3, TCP/IP protocol processing overhead occupies 48.4% of the total overhead. This overhead must be minimized. However, when the network does not support

reliable message transfer at the hardware level, as is the case with Ethernet, message delivery and FIFO-ness must be guaranteed. Therefore the *Go-Back N for PM* protocol proposed in section 4.3 is used to guarantee them.

5.3.3 The Interrupt Reaping Technique

When a parallel application, especially a data parallel application, waits for a message, the message must be processed as soon as possible when it is received. However, in the traditional kernel, when the application issues a primitive to receive a message but no message has arrived, the application is suspended to wait for a message.

Figure 5.2 (a) shows sequences of existing message handling techniques using a hardware interruption upon message reception. The sequences are as follows:

- 1 When a message comes, the NIC hardware triggers a hardware interruption after finishing the message transfer by DMA and setting a finish flag to a descriptor(or register).
- 2 The interruption processing is handled on the host kernel after the hardware interrupt. This processing consists of interrupt control, device driver entry search, etc. This takes 5.9 μ sec as described in Table 5.3.
- 3 Device driver processing is executed and handles received messages.
- 4 Then host user processing handles the message.

The *Interrupt Reaping* technique is proposed to avoid hardware interrupt overhead. In *Interrupt Reaping*, the handler for the device driver is executed by the user program directly when an application is going to wait for a message (Figure 5.2 (b)).

Figure 5.2 (b) shows the message handling sequences using this *Interrupt Reaping* technique on message reception. The sequences are as follows:

- 1 When a message comes, the NIC hardware triggers a hardware interrupt after finishing the message transfer by DMA and setting a finish flag to a descriptor(or register).

- 2 A user program processes a message receive primitive and calls device driver processing using a system call.
- 3 When a message has arrived and the device driver receives it, the handler turns off the interrupt register so that the hardware interrupt will not be activated.
- 4 Then host user processing handles the message.

When a message comes while the user program is not processing a message receive primitive, the message is handled by a hardware interrupt, the same as in Figure 5.2 (a).

Since the *Interrupt Reaping* technique is only used when an application program waits for a message and has nothing to do, the overhead of calling device driver processing using a system call is not considered crucial.

5.3.4 Cluster Communication and Existing Network Protocols

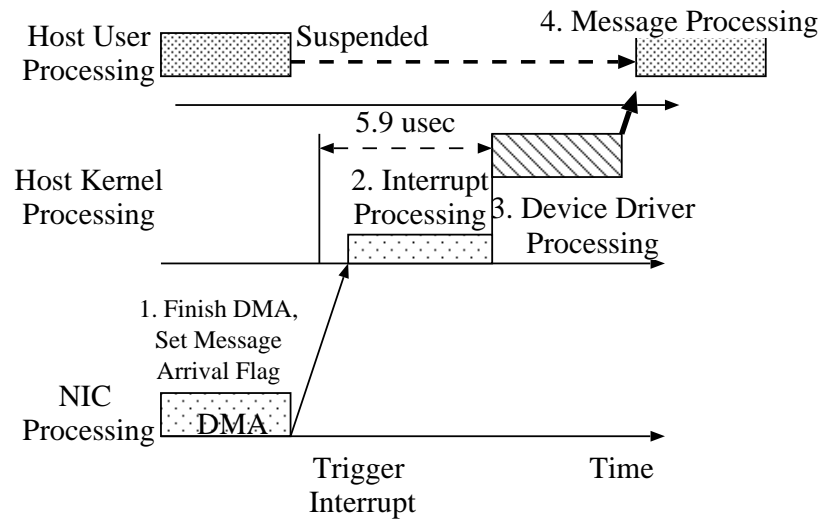
To realize the coexistence of a cluster communication protocol with other existing protocols, the cluster communication uses a special type of packet, and the data link de-multiplexes protocols for cluster and existing communication.

As described in subsection 5.3.1, the cluster communication protocol is handled on the network data link layer. So, message handling sequences on message reception are as follows: when the data link layer receives a packet whose type is that of the cluster communication protocol, the data link layer handles the packet directly. And, when the data link layer receives a packet whose type is different from that of the cluster communication protocol, the data link layer enqueues the packet to an appropriate protocol queue, and triggers a *software interrupt* to process the packet on the upper protocol layer.

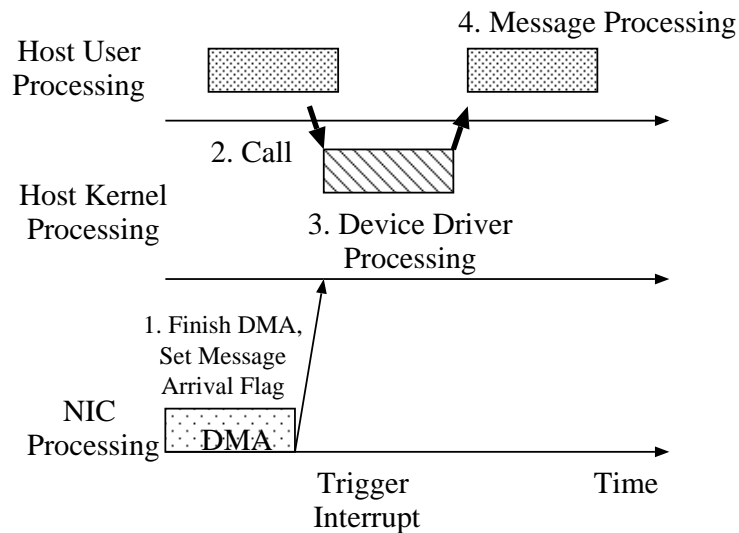
5.3.5 Interrupt Reaping and Existing Network Protocols

The *Interrupt Reaping* technique as described in subsection 5.3.3 is effective in cluster communication. However, the technique has a scheduling issue between processes which use existing network protocols and processes which use a cluster communication protocol in an environment where existing network protocols are also used.

To solve this scheduling issue, the following methods should be used.



(a) Normal Message Handling



(b) Message Handling using Interrupt Reaping

Figure 5.2: The Interrupt Reaping Technique

The process which processes the technique:

- Must not suspend the process while disabling interrupt in order to encourage other interrupts.
- Must not suspend the process in kernel until a message comes in order to encourage

context switching to other processes.

- Should limit the use of the technique in a specified time period in order to increase efficiency.

As described in subsections 5.3.3 and 5.3.4, the interrupt handler used in the *Interrupt Reaping* technique handles not only the packets of cluster communication protocols but also the packets of existing network protocols. So, packets of existing network protocols can be handled in an upper protocol using a *software interrupt*, and other processes which use the cluster communication protocol or an existing network protocol can coexist.

5.4 Implementation of PM/Ethernet

PM/Ethernet is an implementation of the scheme proposed in Section 5.3.

5.4.1 PM/Ethernet Architecture

PM/Ethernet provides same APIs of PM on Myrinet and GigaE PM.

As shown in Figure 5.3, PM/Ethernet consists of a user level library, the PM/Ethernet protocol handling routine, a protocol dispatcher in the Ethernet data link layer, and ordinary Ethernet drivers.

At message sending using the PM/Ethernet library, the PM/Ethernet protocol handling routine assembles an Ethernet frame and calls the Ethernet device driver to send it.

Upon message reception, an arrival message is received by the device driver and then the dispatcher in the Ethernet data link layer calls the PM/Ethernet protocol handling routine if the message type is PM/Ethernet, otherwise it invokes the existing protocol handler.

5.4.2 Implementation on Linux

PM/Ethernet has been implemented on the Linux operating system, modifying operating system kernel sources. Most of the PM/Ethernet code is realized as a Linux device driver, but some modifications are applied to the original Linux kernel sources. The following are modifications to the Linux 2.2.12 kernel source code.

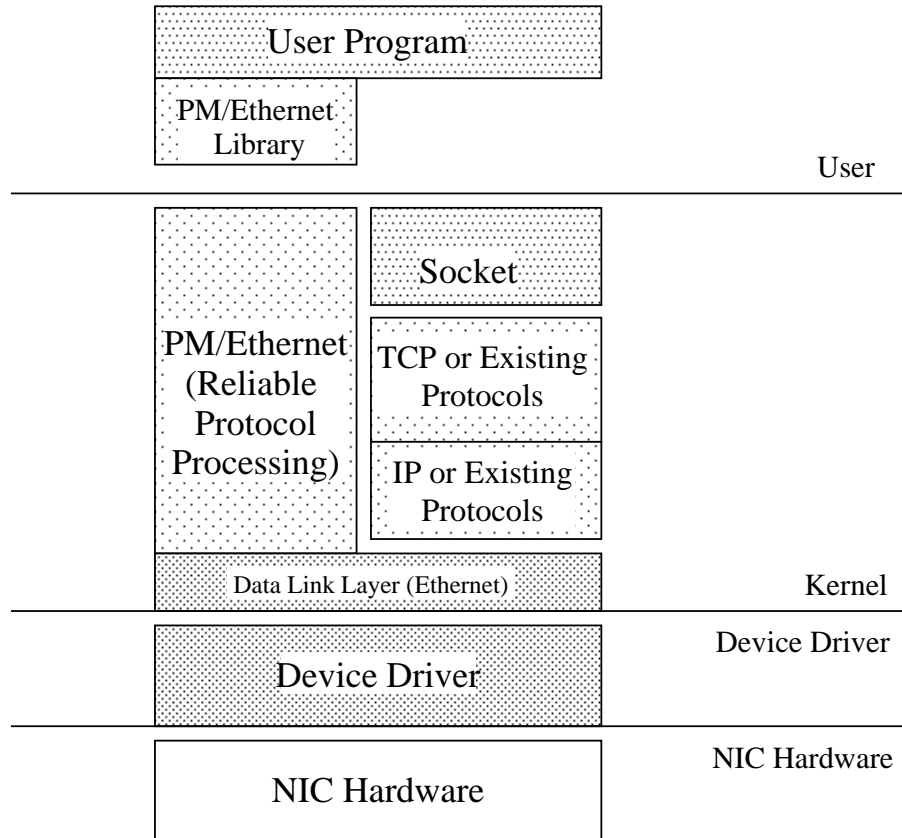


Figure 5.3: The PM/Ethernet Architecture

- The device interrupt handler is registered in the network device driver structure so that it is called by a kernel primitive.
- A dispatcher for the PM/Ethernet Ethernet frame is added in the data link layer.
- A PM/Ethernet device driver is added to operate the PM/Ethernet communication protocol and user interface. This driver also supports dynamic device driver loading.

The following is an overview of PM/Ethernet:

- Send and receive message buffers are pre-allocated in host memory, always pinned down to prevent swapping, and also mapped into user space using the `mmap()` system call.

- At message sending, a user program gets a send message buffer using the `pmGetSendBuffer()` function, it allocates the buffer from the buffer pool mapped to user memory space, and the user program writes data to the buffer. Then the user sends the message using the `pmSend()` function. The PM/Ethernet driver in the Linux kernel builds a header which includes an Ethernet frame and a PM/Ethernet header, and calls the transmit message handler of the Ethernet device handler. In these sequences, there is no data copy performed.
- At message receiving, when the data link layer receives a packet whose type is the cluster communication protocol, the data link layer calls the PM/Ethernet message handler directly, processes cluster communication, and copies the message data from the message buffer of Linux to the receive buffer.

5.4.3 Implementation of Interrupt Reaping

Message reception on PM/Ethernet is implemented using the following code.

```
// on message receive processing in ioctl() function.
if(ret = epm_get_message(channel) == NO_MESSAGE) {
    disable_interrupt();    // disable interrupt
    do_interrupt_handler(); // execute handler
    enable_interrupt();     // enable interrupt
    ret = epm_get_message(channel);
}
return ret;
```

The `do_interrupt_handler()` calls a device driver interrupt handler. Upon message reception, a user program calls this device driver function using a system call. When a user program wants to wait until a message is coming, the user program needs to execute wait loops.

5.4.4 Multi processes and SMP Support

On the Linux version 2.2.12 kernel, only one process can execute the `ioctl` system call by locking on the kernel entry, so there is no need for locking.

However, on an SMP system, a locking mechanism is needed between processes which execute *Interrupt Reaping* and interrupt processing caused by hardware interrupts. To provide this mechanism, PM/Ethernet replaces the original interrupt handler of the device driver with the `do_interrupt_handler()` described below. This code shows that a process does nothing while the other process executes interrupt processing.

```
do_interrupt_handler() {
    if (test_and_set_bit(0, &intr_flags) != 0) return;
    do_original_interrupt_handler();
    clear_bit(0, &intr_flags);
}
```

5.5 Evaluation

5.5.1 Basic Application Level Communication Performance

The basic application level communication bandwidth and latency for PM/Ethernet are measured and compared with that of TCP/IP and PM on Myrinet[THIS97] in this subsection. Table 5.1 shows the evaluation environment. A Gigabit Ethernet switch is not used in this evaluation. The `netperf-2.1pl3`[NET] benchmark program for TCP/IP performance measurement was used.

The bandwidth is shown in Figure 5.4. In Figure 5.4, IR means *Interrupt Reaping*. The PM/Ethernet achieves 77.5 MB/sec of bandwidth with a 1,468 byte message. In contrast to the PM/Ethernet, TCP/IP achieves 46.7 MB/sec of bandwidth with a 1,280 byte message. The communication bandwidth of PM/Ethernet is 1.6 times faster than that of TCP/IP.

PM/Ethernet without *Interrupt Reaping* achieved 64.6 MB/sec of bandwidth with a 1,468 byte message.

Figure 5.5 shows the round trip time. PM/Ethernet achieved a 37.6 μ sec round trip time with a four byte user message. The round trip time on TCP/IP was 89.6 μ sec while the

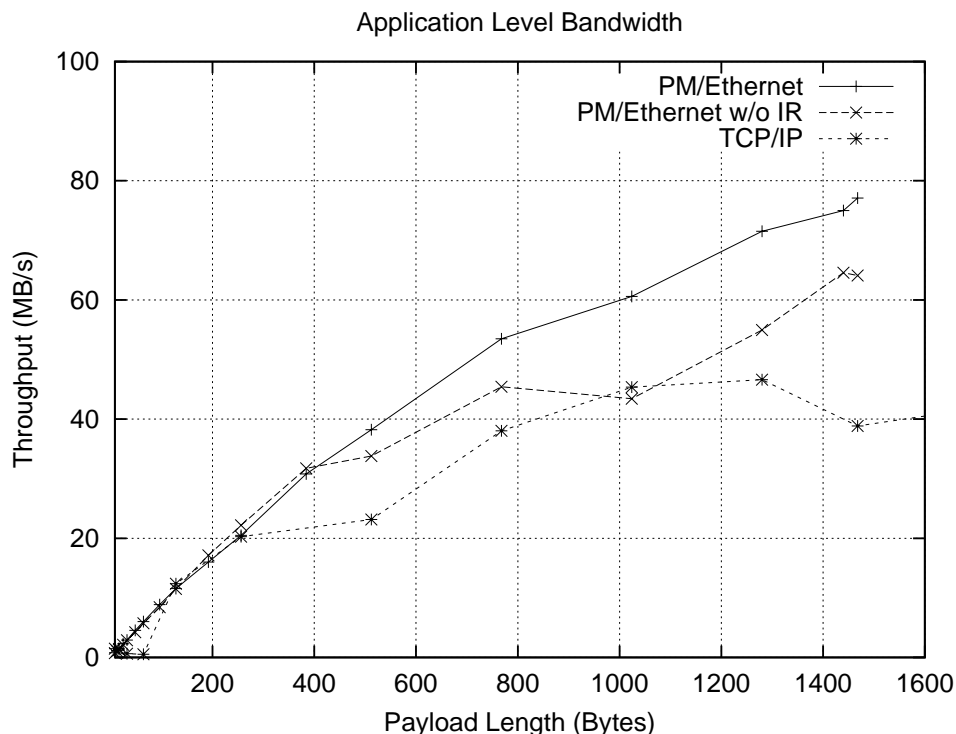


Figure 5.4: Application Level Bandwidth

time on PM/Ethernet was $37.6 \mu\text{sec}$. PM/Ethernet without *Interrupt Reaping* achieved a $47.6 \mu\text{sec}$ round trip time. The communication latency of PM/Ethernet is about one third of that of TCP/IP.

5.5.2 NAS Parallel Benchmarks (NPB)

NPB IS Class A on a Uni-processor Cluster:

Figure 5.6 shows the results of using the NAS parallel benchmarks (Appendix C). version 2.3, IS Class A. All NAS parallel benchmarks have been executed. The LAM MPI implementation [LAM] was used to measure MPI on TCP/IP. The 3Com Super Stack II 9300 Gigabit Ethernet switch was used on this benchmark.

MPI on top of PM/Ethernet achieves an 11.8 fold speedup for IS on 16 nodes. MPI on top of PM/Ethernet without *Interrupt Reaping* achieves an 11.2 fold speedup for IS on 16 nodes. MPI on top of TCP/IP achieves a 5.2 fold speedup for IS on 16 nodes.

The results show that the IS performance on PM/Ethernet is 1.75 times faster than that

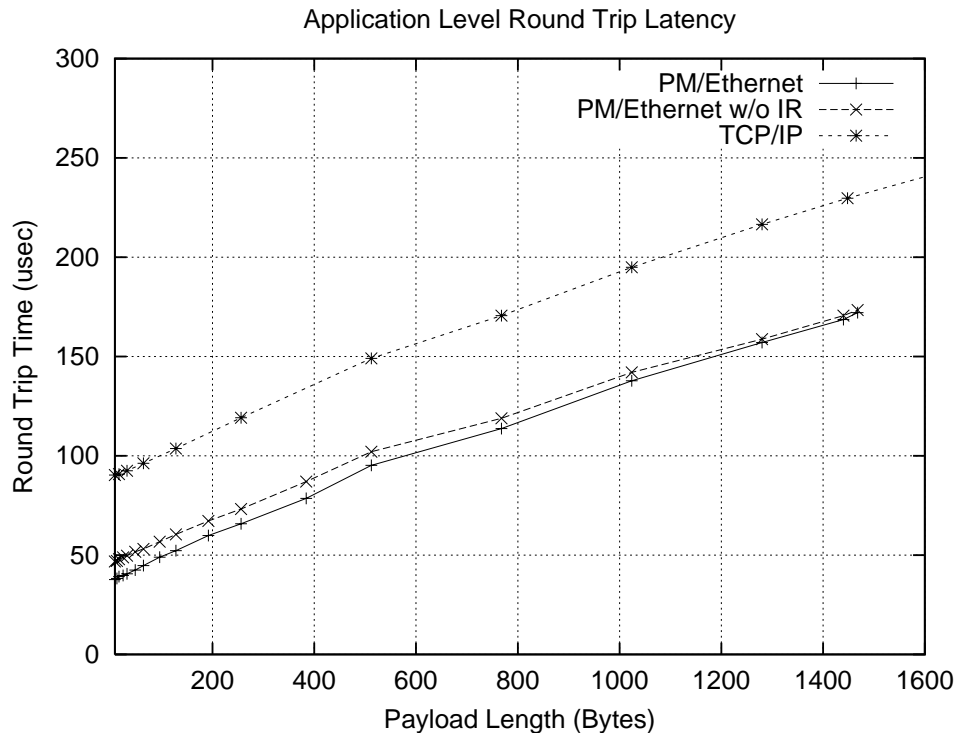


Figure 5.5: Application Level Round Trip Time

on TCP/IP. Because the IS benchmarks are latency and bandwidth sensitive (Appendix C), these results reflect the low latency and high bandwidth of PM/Ethernet.

NPB LU Class A on an SMP Cluster:

Figure 5.7 shows the results of using NAS parallel benchmarks version 2.3 LU Class A on a Dual Pentium III 500MHz 16 node SMP Cluster, and compares the performance of PM/Ethernet on Gigabit Ethernet and Fast Ethernet with the *Interrupt Reaping* technique. The NIC of Fast Ethernet is an Intel EEPRO100, and the NIC is not only used for cluster communication but also used for IP communication, such as NFS and remote shell program for process dispatching. In the legend of Figure 5.7, SMP means two processes are running on each node, the GE means Gigabit Ethernet, and the FE means Fast Ethernet.

Figure 5.7 shows that the PM/Ethernet with the *Interrupt Reaping* technique can run on an SMP cluster, and that cluster communication and the existing IP protocol can coexist.

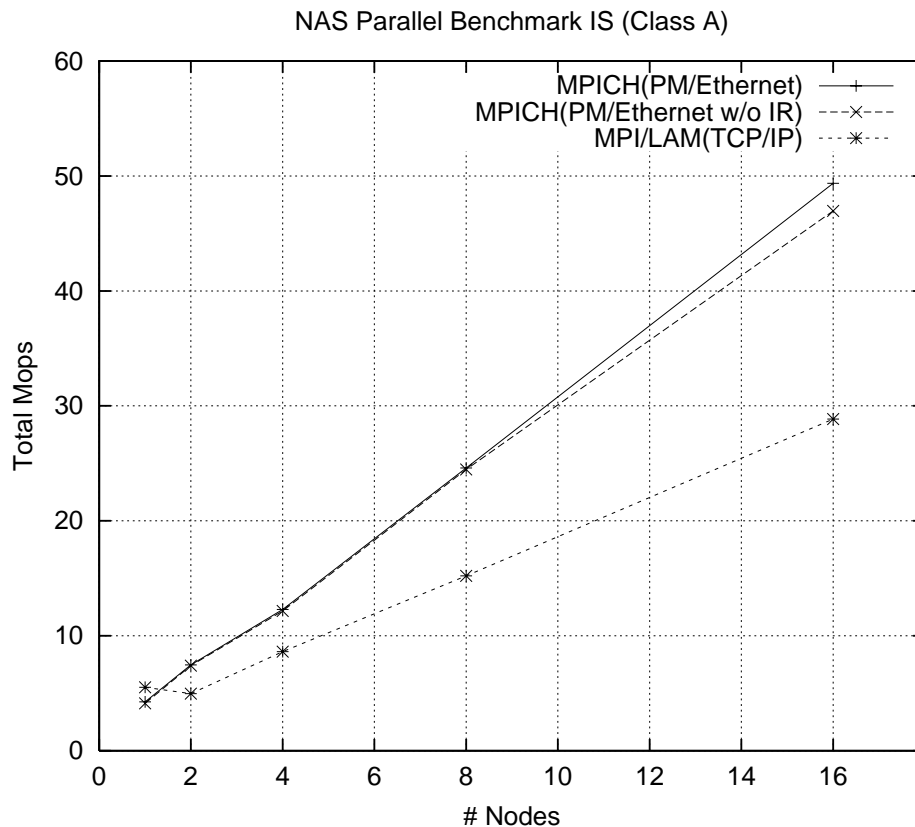


Figure 5.6: NPB IS (Class A) on PM/Ethernet

5.5.3 The Effect of the Interrupt Reaping Technique

As described in subsection 5.5.1, PM/Ethernet achieved a $37.6 \mu\text{sec}$ round trip time with a four byte user message with *Interrupt Reaping*, and without *Interrupt Reaping* it achieved a $47.6 \mu\text{sec}$ round trip time. This result shows that *Interrupt Reaping* eliminated $10.0 \mu\text{sec}$ of hardware interrupt overhead in round trip time. The difference of $0.9 \mu\text{sec}$ between the hardware interrupt cost of $5.9 \mu\text{sec}$ from Table 5.3 and the cost of $5.0 \mu\text{sec}$, which is eliminated by *Interrupt Reaping*, is the cost to call the hardware device interrupt handler in $1/2$ of the round trip time.

Also, as described in subsection 5.5.2, IS Class A performance on PM/Ethernet with *Interrupt Reaping* is 5 % faster than that of PM/Ethernet without *Interrupt Reaping*.

Table 5.5 shows a comparison of the number of interrupts with (and without) the *Interrupt Reaping* technique. These are a sum of the count on all nodes and are calculated from

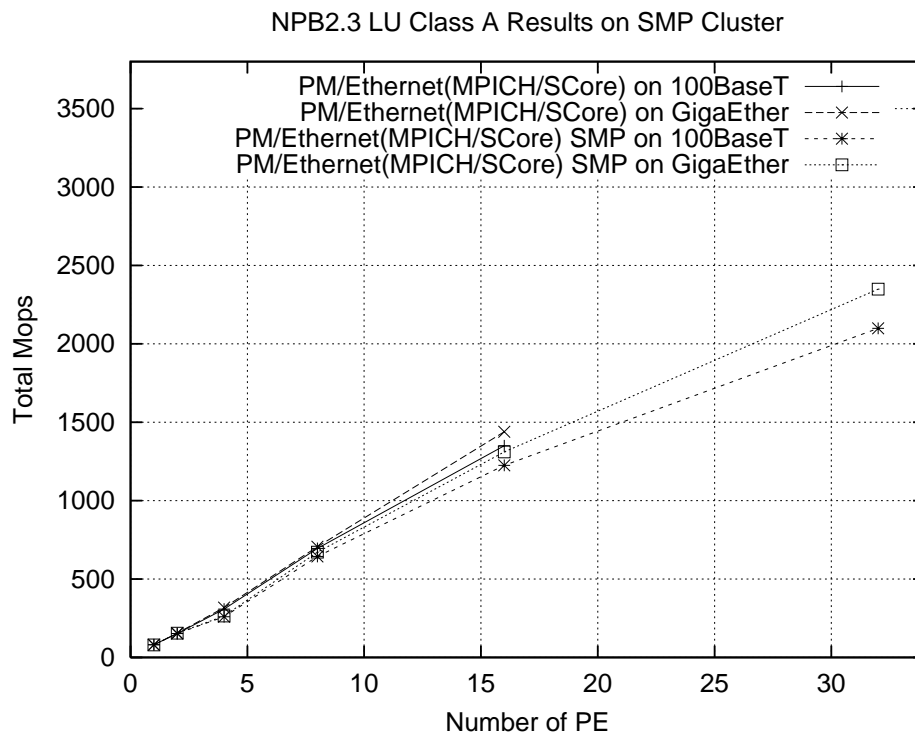


Figure 5.7: NPB LU (Class A) on an SMP Cluster

Table 5.4: Number of Interrupts on IS

IS	Interrupt Reaping		Improvement
	With	Without	
4 PE	70492	92235	23.6%
16 PE	132599	145818	9.1%

the results of the `/proc/interrupts`. Table 5.4 shows that the *Interrupt Reaping* technique reduces the number of interrupts on a real MPI application.

Table 5.5 shows the *Interrupt Reaping* effects on other platforms, a G-NIC II on an Alpha 21264 500MHz CPU with a 64bit PCI (Compaq XP1000), and an Intel PRO/1000 Gigabit Server Adapter (E1000), and an Intel EEPRO100 (100BaseT) on an Intel Pentium III 500MHz PC. IR means *Interrupt Reaping*. Table 5.5 shows that the *Interrupt Reaping* technique is effective on other platforms.

Table 5.5: Interrupt Reaping on Other Platforms

Processor, NIC	RTT w/ IR	RTT w/o IR	Bandwidth w/ IR	Bandwidth w/o IR
Alpha 21264, G-NIC II	44.6 μ sec	49.8 μ sec	96.7 MB/s	82.5 MB/s
Pentium III, Intel E1000	70.1 μ sec	108.0 μ sec	71.0 MB/s	71.2 MB/s
Pentium III, EEPRO100	106.4 μ sec	128.2 μ sec	11.9 MB/s	11.8 MB/s

5.5.4 PM/Ethernet Protocol Processing Cost Analysis

Table 5.6 shows a comparison of protocol processing cost analyses between TCP/IP and PM/Ethernet on 1/2 of the round trip time. The evaluation environment is the same as that of Table 5.1.

Table 5.6: Comparison of Protocol Processing Cost Analysis on 1/2 of the Round Trip Time

Processing	TCP/IP	PM/Ethernet
User Program, System Call and Socket	1.6 μ sec	1.6 μ sec
TCP	15.5 μ sec	-
IP	6.2 μ sec	-
Light-weight Protocol	-	4.8 μ sec
Protocol Handler Invocation	3.2 μ sec	-
Device Driver	4.7 μ sec	4.7 μ sec
Hardware Interrupt	5.9 μ sec	-
NIC+Media	7.7 μ sec	7.7 μ sec
Total	44.8 μ sec	18.8 μ sec

PM/Ethernet eliminates 77.9% of the communication protocol cost compared with TCP/IP using a light weight communication protocol, and eliminates the costs of protocol context switching and hardware interrupts using the *Interrupt Reaping* technique.

5.6 Related Work of PM/Ethernet

U-NET[BBVvE97] also achieves low latency and high bandwidth communication using user level communication. Fast Traps using a trap gate in U-NET/MM[MW97] is only useful for Intel machines to reduce the overhead of system calls. The *Interrupt Reaping* technique is CPU architecture independent.

GAMMA[CC00] achieves low latency and high bandwidth communication using kernel level communication and 100 Mbps Ethernet, and also provides existing network protocols, such as TCP/IP. However, GAMMA does not provide reliable communication, so a program fails when a message is lost. And, the implementation of GAMMA is dependent on NIC device drivers.

AM[T. 92], AM-II[CMC97], GM[GM], FM[Sco95], BIP[PT98], VMMC-2[DBC⁺97] and PM[THIS97] are based on Myrinet, and achieve low latency and high bandwidth communication using user level communication and firmware on a Myrinet NIC. VIA[VIA], the Virtual Interface Architecture, is being widely implemented in Gigabit class networks on the Microsoft Windows operating system.

All of these communication facilities have hardware device dependent code or CPU architecture specific code. However, PM/Ethernet is NIC hardware independent.

5.7 Conclusions of This Chapter

This chapter has proposed software techniques to realize a high performance communication facility using a commodity network. These techniques do not require any special hardware or hardware specific device drivers in order to adapt to any kind of NIC. In these techniques, a reliable light-weight network protocol is handled on the data link layer called by a network device driver directly, and existing network protocols such as TCP/IP are also supported on the same NIC. The *Interrupt Reaping* technique is proposed to eliminate the hardware interrupt overhead when an application waits for a message.

PM/Ethernet, an instance of this scheme, is implemented on Linux with minimal modification to the Linux kernel, and existing network device drivers are used without any modification. It achieves 77.5 MB/s bandwidth and 37.6 μ sec round trip time latency

compared to the fact that TCP/IP achieves 46.7 MB/s bandwidth and 89.6 μ sec round trip time latency using Pentium III 500 MHz PCs on Packet Engines' G-NIC II Gigabit Ethernet NIC. The communication latency of PM/Ethernet is about one third of that of TCP/IP, and the communication bandwidth of PM/Ethernet is 1.6 times faster than that of TCP/IP. Using the NAS parallel benchmark programs, MPI on PM/Ethernet was evaluated. The results show that MPI on PM/Ethernet achieves 1.75 times faster than MPI on TCP/IP. Using the *Interrupt Reaping* technique, execution speed for applications is 6 % faster than not using the technique.

Chapter 6

Software Techniques Using Multiple NICs

In this chapter, the *Network Trunking* technique is proposed in order to achieve high communication bandwidth by combining multiple NICs into one big channel. Then the communication performance of this technique is evaluated. As for similar communication techniques, the *Channel Bonding*[T. 95] technique used by Beowulf was proposed, but there are some problems with utilizing it on a cluster network for parallel computing. This chapter points out the problems and proposes a method which solved problems found with the *Channel Bonding* technique of Beowulf.

6.1 Network Trunking Design

6.1.1 Characteristics of Ethernet Switch

Recently, many Ethernet switches have a table to memorize the Ethernet MAC address of each host which is connected to every switch port. This table is built from the MAC address of the source sending an Ethernet Frame arriving at each port.

When an Ethernet frame comes to an Ethernet switch, the Ethernet switch searches for the destination MAC address of the Ethernet frame in this table. When the address is found in the table, the Ethernet frame is sent to the registered entry of the table. When it is not found, the Ethernet switch broadcasts the Ethernet frame to all ports of the Ethernet switch.

By memorizing MAC addresses, the Ethernet switch increases the efficiency of transmitting Ethernet frames, eliminating redundant frame transmission.

When Ethernet frames with the same source MAC address come from multiple ports of an Ethernet switch, the transmitting process depends on the implementation of the switch. Some switches broadcast the Ethernet frame to all ports, some switches send to multiple ports which are then registered in the table.

6.1.2 The Beowulf Channel Bonding Technique

The *Channel Bonding*[T. 95] technique of Beowulf does not maintain the Ethernet MAC addresses of its own NICs and the destination Ethernet MAC addresses, so it is not able to convert Ethernet frames to the appropriate destination MAC address. The *Channel Bonding* technique also requires that every NIC needs to be connected to the same Ethernet network.

The *Channel Bonding* technique is useful in the case where sending frames on the server NIC become bottlenecked. However, there are the following problems:

- Load distribution is possible on the sending side, but not on the receiving side, since communication concentrates on one specified piece of NIC hardware.
- A switch may broadcast an Ethernet frame when frames with the same source MAC

address come to the switch, as described in section 6.1.1. And there is also the case where all the NICs interfere with the broadcast frames.

- When every NIC can connect with one switch, collective communication performance is limited by the bisection bandwidth of the switch because every NIC needs to be connected to the same Ethernet Network, or limited in communication performance to the links between switches.

6.1.3 Proposing the Network Trunking Technique

The *Network Trunking* technique solves the problems of the *Channel Bonding* technique by rebuilding the Ethernet frame using the PM/Ethernet mechanism to all of the destination nodes. Features of the *Network Trunking* technique on PM/Ethernet are described as follows:

- Requests for Ethernet switch bandwidth are reduced compared with the *Channel Bonding* technique because each network can be separated using each Ethernet NIC on a host.
- There is no annoying effect to the switch transfer shown above because of restructuring of the Ethernet frame.
- Low communication overhead using the *Go-Back N for PM* protocol proposed in the GigaE PM is used instead of the TCP/IP protocol.

The followings are the design issues of the *Network Trunking* technique on PM/Ethernet.

- 1 Implementation technique to guarantee message arrival and message ordering when multiple NICs are used.
- 2 Implementation of the *Interrupt Reaping* technique in the *Network Trunking* technique.

1. Implementation technique to guarantee message arrival and message ordering when multiple NICs are used

Message order is easily changed using multiple NICs. Accordingly, a guarantee of message arrival and message ordering is required. However it had better be able to queue a mis-ordered message as well, because a proper message may arrive at the other NICs.

So a message ordering queue is introduced to buffer a message by mean of every context of each PM. The queue is in the PM context structure that is the unit of message administration of PM.

When a message does not arrive, it can not be judged whether the message transfer is delayed or the message is really lost. This problem is solved by using a re-transmission algorithm that re-transmits when an ACK message is not received by the sending side after a fixed periods. Redundant messages are discarded on the receiver side.

2. Implementation of the *Interrupt Reaping* technique in the *Network Trunking* technique

It is not enough to execute *Interrupt Reaping* processing for a single device. It must be done for all devices.

6.2 Network Trunking Facility Implementation

The *Network Trunking* facility is implemented using the PM/Ethernet mechanism on the Linux operating system. The user library and PM/Ethernet device driver were modified to accommodate PM/Ethernet.

- User library: only a part of the initialization in Network Trunking correspondence is expanded.
- PM/Ethernet driver: the *Network Trunking* part is separated from the other parts, such as the sending and receiving processes, that are not related to the *Network Trunking* part to minimize the need for additional code for the *Network Trunking* facility.

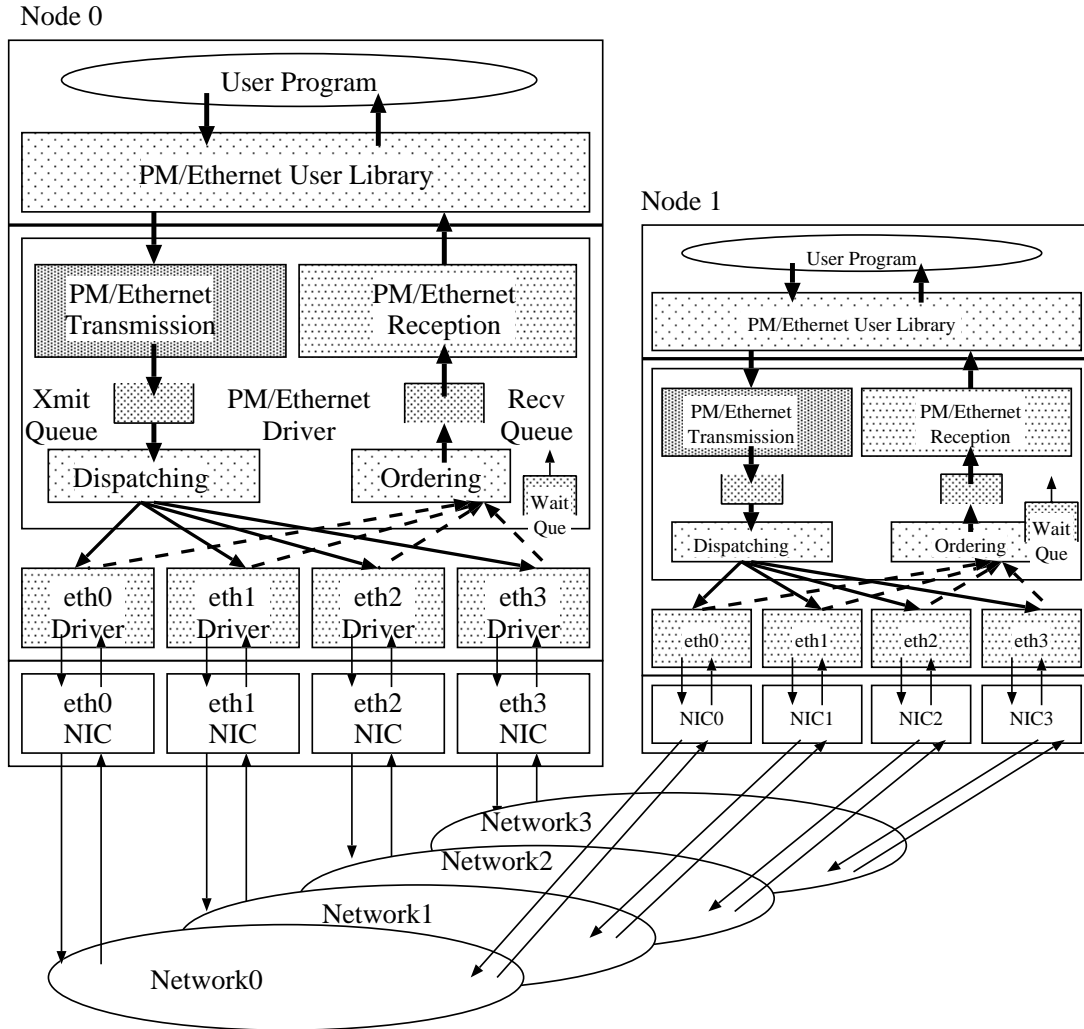


Figure 6.1: The Network Trunking Architecture

Figure 6.1 shows the internal configuration of the *Network Trunking* facility. The *Network Trunking* facility is realized using a part of the lower portion of Dispatching and Ordering from the PM/Ethernet driver described in Figure 6.1, and these parts control multiple Ethernet device drivers.

Dispatching: This part has a table consisting of the MAC address of every node used by the NICs(eth0,1,2,3). After having decided on a transfer NIC according to the message from the sending queue, the Dispatching part replaces the Ethernet header, and calls the sending function of the Ethernet device driver.

For communication among many nodes, the round-robin method was adopted in order to utilize all of the NICs equally.

Ordering: Message ordering is managed by sequence number. A message is stored in the reception queue when the sequence number matches during reception and inserted in the internal wait queue when the sequence number does not match.

The ordering part always checks whether the sequence number matches or not during reception of a message. When the sequence number of a message matches, the message is moved to the receive queue from the wait queue.

When the message has already been received in the wait queue, the message is discarded.

Normal device driver structure (`netdevice` structure) is used in controlling multiple Ethernet device drivers in order to maintain the independence of existing Ethernet device drivers.

The *Network Trunking* facility is implemented without any modification to existing Ethernet device drivers. So, there is no need to use the same NIC hardware.

6.3 Evaluation of Network Trunking

6.3.1 PM level Communication Performance

In this section, the PM level communication performance of the *Network Trunking* technique is evaluated.

Table. 6.1 shows the evaluation environment. The Digital DC21140(Tulip), Intel EEP-RO100 and 3Com 3C905B were selected as evaluation NICs because they are currently the most popular NICs. The device driver for all three Ethernet NICs was written by the same developer, and the control code of each device driver is similar to that of the others, except for the initialization of the device.

Figures 6.2, 6.3 and 6.4 show the PM level communication bandwidth achieved by changing the number of NICs for the Tulip, EEP-RO100 and 3C905B.

Table 6.1: Evaluation Environment for Network Trunking

Hardware	Pentium II 400MHz (440BX chipset, 256MB SDRAM)
NIC	Digital DC21140(Tulip) Intel EEPRO100 3Com 3C905B using 32 bit PCI slot
Host OS	Redhat 6.1 Linux (2.2.12 kernel)
Device Driver for Tulip for EEPRO100 for 3C905B	tulip.c:v0.91g-ppc 7/16/99 Donald Becker eeepro100.c:v1.08 5/3/99 Donald Becker 3c59x.c:v0.99H 11/17/98 Donald Becker
Ethernet Switch	Extreme Summit 2

The results showed improvement of communication bandwidth to be almost linear until three NICs are used. This holds for all three types of NICs. The best communication performance with four NICs was 44.5 MB/s with the Tulip (3.6 times faster than one NIC). In addition, the performance of the 3C905B with 4 NICs was 29.8 MB/s, but this was worse than that of 3 NICs.

Table 6.2 shows the round trip time for a 4 byte message using each NIC. The round trip time of the Tulip increased by 1-2 μ sec by increasing the number of NICs, however the round trip time of the EEPRO100 and the 3C905B increased by 6 μ sec. Because the execution code of PM/Ethernet is the same, it is thought this is due to a problem of the device driver or some problem on the NIC hardware side.

Table 6.2: Round Trip Time of Network Trunking (μ sec)

NIC Hardware	NIC \times 1	NIC \times 2	NIC \times 3	NIC \times 4
Tulip	84.5	85.6	87.6	88.9
EEPRO100	97.6	103.6	105.6	106.0
3C905B	89.8	95.8	97.9	100.4

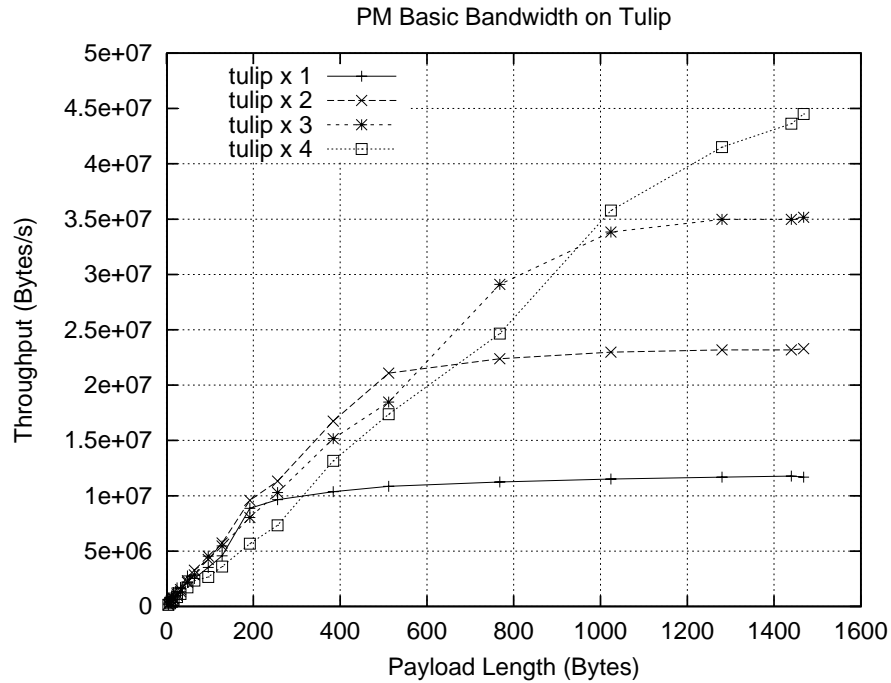


Figure 6.2: Communication Bandwidth on the Digital Tulip

6.4 Conclusions of This Chapter

This chapter proposed and evaluated the *Network Trunking* technique, which allows communication bandwidth to improve by using multiple NICs.

The *Network Trunking* facility improved the communication bandwidth using up to three NICs for all three kinds of NICs (Tulip, EEPRO100, 3C905B). The best communication performance at four NICs was 44.5 MB/s with the Tulip (3.6 times faster than using one NIC). The *Network Trunking* facility enables a great deal of bandwidth improvement when using a method that does not depend on hardware.

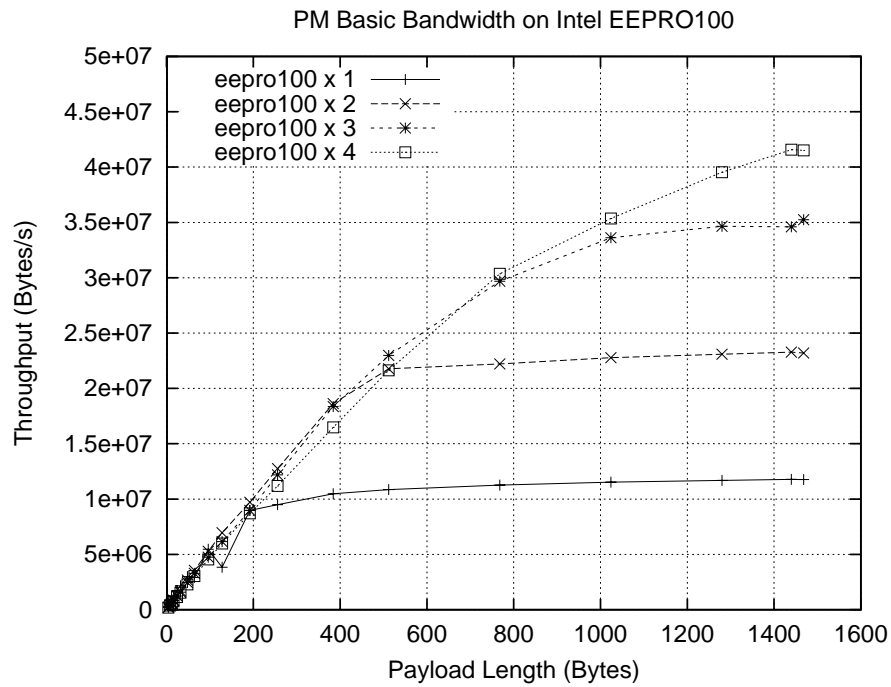


Figure 6.3: Communication Bandwidth on the Intel EEPRO100

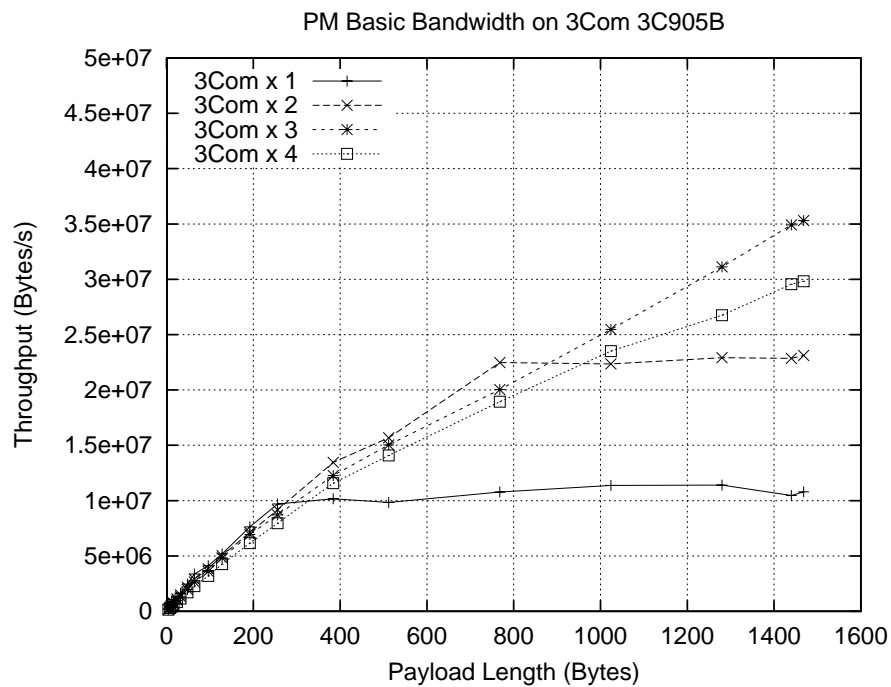


Figure 6.4: Communication Bandwidth on the 3Com 3C905B

Chapter 7

Comparison of Application Performance with a Dedicated Cluster Network

This chapter shows that the PM/Ethernet on Gigabit Ethernet is able to achieve application performance comparable to the PM/Myrinet as a dedicated cluster network. In order to evaluate performance involving a difference of networks precisely, PMv2 communication facility was used. Application performance on Gigabit Ethernet and Fast Ethernet networks (including *Network Trunking*) is also evaluated, and compared with that on Myrinet. This evaluation is done on RWC clusters: RWC SCore Cluster I, RWC SCore Cluster II and RWC PC Cluster II.

7.1 Cluster of Clusters

PC clusters are already shifting to the full-scale implementation stage in addition to being a target of study as described in chapter 2. With the spread of PC clusters, it is expected that a cluster system network will appear and user requests for a cluster system will be diversified. In addition, as cheap symmetric-multi-processor (SMP) PCs become available, an SMP PC cluster which uses them will attract a lot of attention.

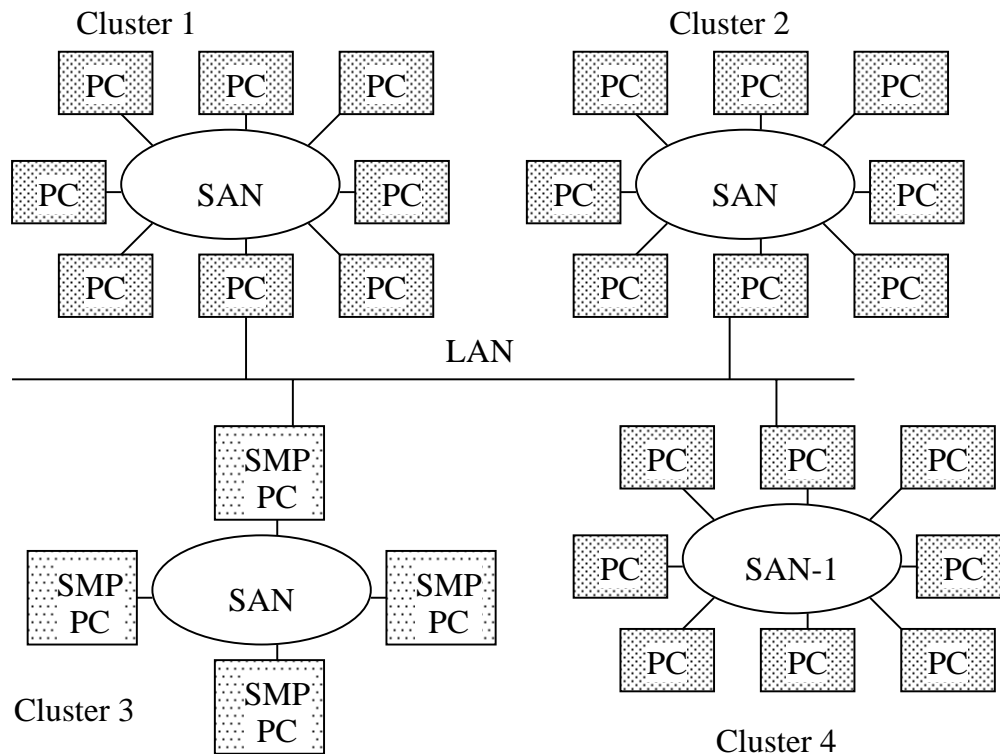


Figure 7.1: Example of a Cluster of Clusters

As one example of this diversification of cluster configuration, users want to utilize more than one cluster as part of a bigger cluster than each of the member clusters. It is effective when large-scale computing performance is necessary in an environment where more than one cluster is used. Such a cluster is called a *COC* (Cluster of Clusters, Figure 7.1). In a *COC*, networks for these clusters may not always be the same type of network, and it is a problem as to how these different types of networks should be treated.

7.2 PMv2 Design

PMv2 is a communication facility designed to support a Cluster of Clusters (*COC*) and an SMP cluster, and to provide communication architecture for more than one network.

7.2.1 *COC* and SMP Cluster Support

Figure 7.2 shows an example of a *COC* consisting of a PC cluster using Gigabit Ethernet, and an SMP PC cluster using Myrinet. Every PC is connected with Fast Ethernet.

It is valid to select the highest-speed network as the network to use for communication. In cluster 1, Myrinet should be used for communications between nodes, and shared memory communication should be used for the interprocess communication (Figure 7.2, Shmem). In cluster 2, Gigabit Ethernet should be used between the nodes. And Fast Ethernet is only available for communication between nodes in cluster 1 and cluster 2.

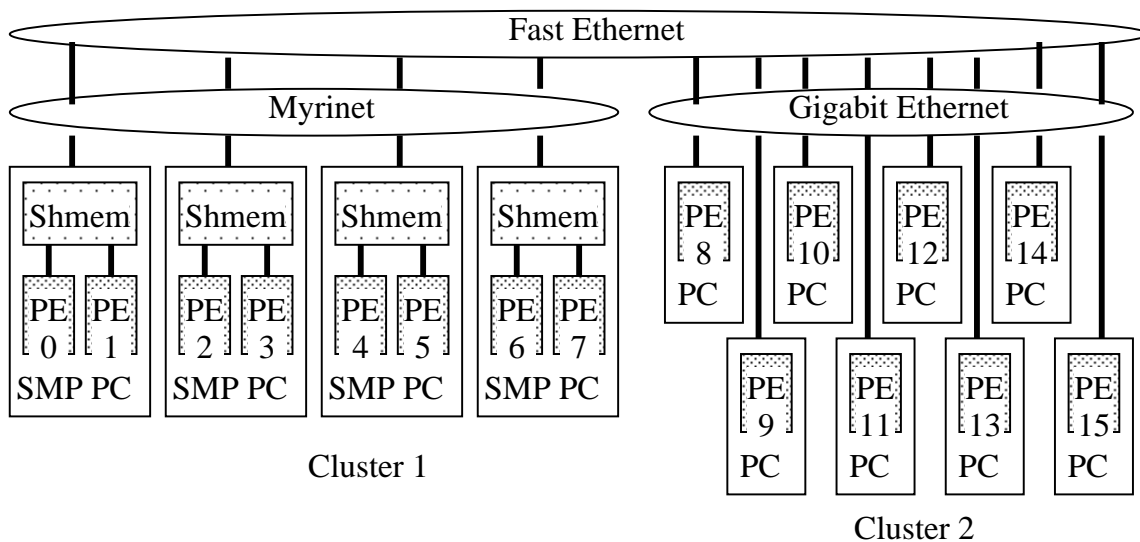


Figure 7.2: Example of a Cluster of Clusters

In communication of *COC*, it is natural that interprocess communication between PEs on an SMP PC is considered as a kind of network communication as shown in Figure 7.2. Therefore, it is good to provide a framework to consider shared communication between PEs on the SMP PC as a kind of communication.

To provide this framework, a mechanism called *PM/Composite* has been proposed[HTT⁺99]. In *PM/Composite*, each node has a destination PM device table that stores what network should be used for each destination node, and when a message sending request to a node is posted to *PM/Composite*, the *PM/Composite* gets the destination network from the destination PM device table for the node and sends the message to the network.

For example, Table 7.1 shows a destination PM device table located in PE0 of Figure 7.2. In addition to the above, to make a binary program work in a different network environment, the PM device table for the appropriate network should be made available at the start of the program.

Table 7.1: An Example of a Destination PM Device Table on PM/Composite

PE No.	Device	PE No.	Device	PE No.	Device	PE No.	Device
0	-	1	SH	2	MY	3	MY
4	MY	5	MY	6	MY	7	MY
8	FE	9	FE	10	FE	11	FE
12	FE	13	FE	14	FE	15	FE

SH=Shmem, MY=Myrinet, FE=Fast Ethernet

7.2.2 Communication Architecture for Multiple Networks

An example of a typical communication architecture that supports multiple networks is the protocol stack of a UNIX based OS (Figure 7.3). A protocol stack controls the communication protocol between the device driver which controls the hardware and the user program.

An advantage of the protocol stack of the UNIX based OS is to provide independence between multiple devices and multiple protocols by prescribing an interface between the devices and the protocols. This feature increases the applicability to other protocols.

Using the protocol stack of the UNIX based OS, implementation of a communication protocol to enable reliable communication on Ethernet, Myrinet and shared memory is discussed.

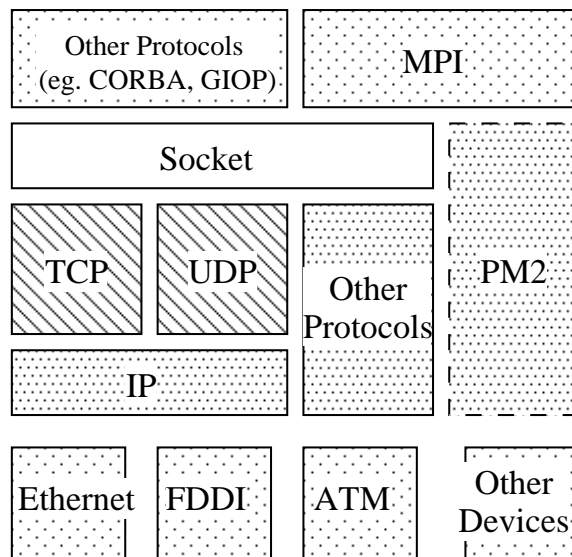


Figure 7.3: Protocol Stacks on the UNIX Operating System

Table 7.2 summarizes the requirements needed for protocol processing to ensure communication with reliability in each network. Because Ethernet does not guarantee message transfer at the hardware level, a message may be lost. Accordingly, protocol processing that guarantees message arrival, message ordering, and flow control of the buffer must be implemented to ensure reliable communication on Ethernet. However, because Myrinet and interprocess communication using shared memory both guarantee message transfer at the hardware level, there is no need of protocol processing to guarantee message arrival and message ordering. It is enough to process the flow control of the buffer.

Table 7.2: Protocol Processing Requirements on Each Network

Network	Guarantee of Message arrival and Message Ordering	Flow Control of Buffer
Shared Memory	-	Required
Myrinet	-	Required
Ethernet	Required	Required

In meeting the requirements shown in Table 7.2 using a framework based on the protocol stack of the UNIX based OS on Myrinet and shared memory, a communication protocol

that guarantees message arrival and message ordering is also processed on both Myrinet and shared memory. This approach increases the protocol processing overhead. So, it is not possible to take advantage of hardware performance when using the protocol stack of the UNIX based OS.

Because it is important in a communication facility for a cluster to maximize the performance of the network hardware, a framework involving the protocol stack on the existing OS cannot be used. A method that is most suitable for each network is chosen.

7.2.3 Support of Hardware Specific Communication

In high performance communication for a cluster, it is important to maximize network hardware performance. Accordingly, a communication method which depends on specialized hardware should be introduced if conspicuous improvement in communication performance is expected.

However, if application program interfaces (APIs) are hardware specific, the method may not be implemented for every type of network hardware. This limits the portability of applications when APIs have to be made individually for every communication method.

PMv2 adopts APIs as options in order to introduce communication methods that make use of specialized hardware possible. A program judges whether these APIs can be used on the program or not by the attributes of a device. If the attributes include the flags of the APIs, then it can be used. To improve the portability of an application, the addition of APIs for optional operations should be minimized and the APIs should be as widely usable as possible.

7.2.4 PMv2 APIs

Table 7.3 shows the essential APIs of PMv2. The APIs of PMv2 consist of those for PM device operations, PM context processing operations, PM message communication operations and PM remote memory operations. In these operations, every PM device must implement APIs for PM device operations, PM context processing and PM message communication.

For details about the APIs on PMv2, refer to the PM Application Programmers' Interface

Manual[PMA].

Table 7.3: APIs on PMv2

PM Device Operations (Required)	Description
pmOpenDevice() pmCloseDevice() pmGetOptionBit()	Open a PM device Close the PM device Get the options of the PM device
PM Context Operation (Required)	Description
pmOpenContext() pmCloseContext() pmAssociateNodes()	Open a PM context Close the PM context Register nodes to the PM context
PM Message Operation (Required)	Description
pmGetSendBuffer() pmSend() pmReceive() pmReleaseReceiveBuffer()	Get a PM send buffer Send a message Receive a message Release a receive buffer
PM Remote Memory Operation (Optional)	Description
pmMLock() pmMUnlock() pmWrite() pmIsWriteDone() pmRead() pmIsReadDone()	Pin-down user memory Release pinned-down user memory Process remote memory write Check completion of remote memory write Process remote memory read Check completion of remote memory read

PM device operation and PM context operation APIs: (Required)

These APIs process opening and closing of PM device and PM context, and initialization and building of a communication environment for the PMv2 communication facility.

PM message communication APIs: (Required)

These APIs process polling-based message communication.

PM remote memory operation APIs: (Optional)

These APIs process pin-down operations of user memory and remote memory access

operations. *Direct Memory Copy* techniques [TOT⁺99] and *Zero-Copy* techniques [TOHI98] are hardware specific communication on PMv2. These techniques use APIs of remote memory operations to be as general purpose as possible.

7.2.5 Implementation Techniques for PM Devices

Implementation techniques depend on the characteristics of each network. For example, the following cases are considered:

- Processing of the PM protocol and the device control can be handled in the NIC CPU when the NIC has its own programmable CPU.
- When there is an existing device driver, the PM protocol can be handled on top of the driver.

7.3 PMv2 Implementation

7.3.1 Overview of the PMv2 Implementation

PMv2 was implemented on Linux. Figure 7.4 shows the PMv2 architecture. PMv2 consists of PM/Myrinet, PM/Ethernet, PM/Shmem and PM/Composite as PM devices in addition to a PMv2 library which controls the whole PMv2 library (Figure 7.4, PMv2 Lib).

As described in section 7.2.2, PMv2 adopts the communication protocol implementation method to maximize the performance of the hardware with Myrinet, Ethernet and shared memory, using a simple communication protocol for the cluster system.

A brief description of the implementation of each PM device is given below:

PM/Myrinet: a communication facility on Myrinet [THI96, THIS97, TOHI98] which is a high-speed network with a bandwidth of 160MB/s (Section 3.2.1).

PM/Ethernet: a communication facility using existing Ethernet device drivers without any modification to the existing Ethernet device driver (Chapter 5) [SHT⁺99a] [SHT⁺00a].

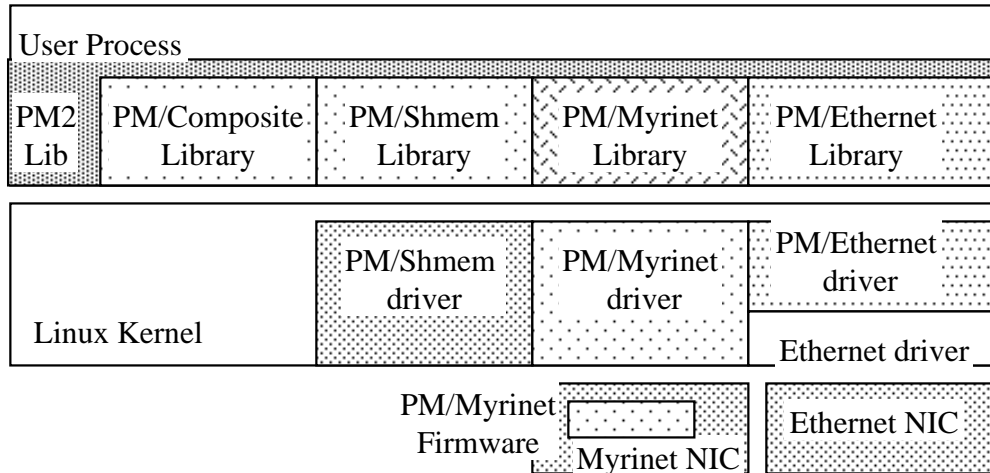


Figure 7.4: PMv2 Architecture

PM/Shmem: a communication facility designed to consider interprocess communication through the shared memory as a PM communication. PM/Shmem is based on the functionality of MPICH-PM/CLUMP[TOT⁺99].

- Shared memory is allocated as PM context and the PM/Shmem library communicates through this shared memory.
- The PM Communication protocol is processed in the PM/Shmem library. Protocol processing is a simple ring buffer process.
- The *Direct Memory Copy* technique is realized in the PM/Shmem device driver.

PM/Composite: this PM device switches the PM device for every partner node.

- APIs for switching networks to every node are added and realized in the PM/Composite library.
- PM/Composite includes an exclusive control mechanism for SMP and the destination PM device table of partner point nodes.
- On a message transmission, the message is transmitted using the PM device corresponding to the sender node in the PM device destination table. And, in message reception, PM/Composite polls every PM device registered in the PM

device destination table.

- The overhead of PM/Composite includes the exclusive control for the SMP, the accessing cost of the table of the PM device information of the partner device, and cost of indirect function calling.

Among the APIs described in section 7.2.4, the PM device operation is implemented in the PMv2 library. Network dependent PM device operations, the PM context operations, and the PM message communication operations are implemented in each PM device independently. The PM remote memory operations are implemented in PM/Myrinet and PM/Shmem.

A pointer to a function corresponding to the APIs is stored in the PM device specific context structure, and APIs for each PM device are executed by an indirect function call.

PMv2 can be developed on hardware which has a programmable CPU on the NIC or which can control a hardware device using a device driver in the kernel.

7.3.2 SCore Version 3 and MPICH/SCore on PMv2

SCore Version 3[HTT⁺99](SCore3) is cluster system software on PMv2. Figure 7.5 shows the architecture of the SCore3 cluster system software. In the execution environment of SCore3, a user specifies a network, the configuration of the network, and the number of nodes with a command option.

The SCore3 runtime environment makes a *PM/Composite* device and executes the program using the *PM/Composite* device. In the case of an SMP cluster, the SCore3 runtime environment makes multiple processes on one node, and makes a *PM/Composite* device for each process. This means that multiple processes share one PM device.

MPICH/SCore is an MPI library in SCore3, based on MPICH-1.1.2[MPIb], and is implemented using the Channel interface of the MPICH.

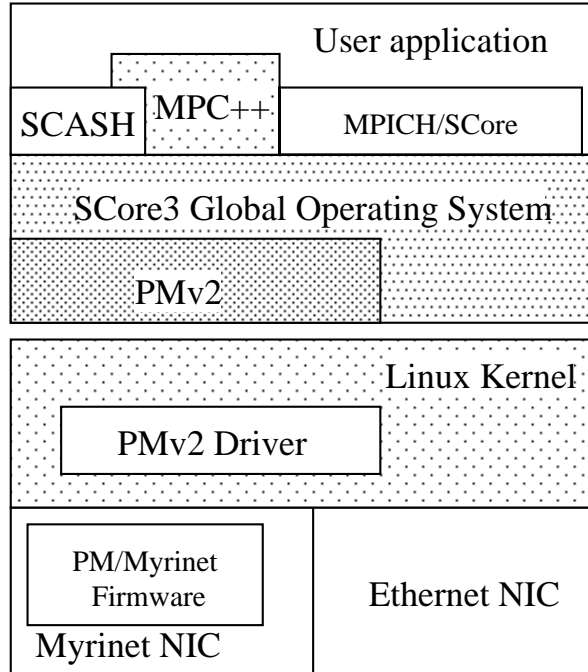


Figure 7.5: SCore3 Architecture

7.4 Performance Evaluation on RWC SCore Cluster I

The following sections (sections 7.4, 7.5 and 7.6) compare the application performance on PM/Ethernet with that on PM/Myrinet.

In the following evaluation, basic communication performance and the NAS parallel benchmarks are measured (Appendix C). TCP/IP is also used as a comparison target for PM/Ethernet and PM/Myrinet. The same binary programs are used as benchmarks, changing the network with the command option at runtime.

The performance with a 16 node SMP Pentium III cluster in a part of SCore Cluster I were measured. The measurement environment is shown in Table 7.4. The network switch shown in Table 7.4 is used in every measurement. In this evaluation, a 32 bit 33 MHz PCI version of Myrinet was used. In PM/Ethernet and TCP/IP, the benchmark programs on Gigabit Ethernet and Fast Ethernet (100BaseT) are measured. In the following measurements, PMv2 message communication is used.

Table 7.4: Measurement Environments on RWC SCore Cluster I

Node PC	DUAL Pentium III 500MHz (440BX chipset, 512MB SDRAM 32 bit 33MHz PCI bus)
Ethernet NIC	Packet Engines G-NIC II(Gigabit Ethernet) Intel EEPRO/100(100BaseT)
Ethernet Switch	3Com SuperStackII 9300(Gigabit Ethernet) 3Com SuperStackII 3900(100BaseT)
Myrinet	Myricom M2M-PCI32 (32 bit PCI) 33 MHz LANai4 processor with 1 MB memory
Myrinet Switch	Myricom M2M-OCT-SW8
Host OS	Redhat 6.1 Linux (2.2.12 kernel)
G-NIC II Device Driver	hamachi.c:v0.11 8/21/99 Written by Donald Becker
EEPRO/100 Device Driver	eeepro100.c:v1.09j-t 9/29/99 Written by Donald Becker

7.4.1 Basic Communication Performance on PM/Myrinet, P-M/Ethernet

In this section, the PM level bandwidth and round trip time are compared with TCP/IP as a measure of basic communication performance. The communication performance of TCP/IP using Netperf-2.1pl3 [NET] was measured with the NODELAY option.

Bandwidth performance was calculated from the time to transmit messages 100,000 times consecutively, and round trip time is an average value calculated from the time it took to execute 100,000 times using a ping-pong program. On every measurement, Ethernet and Myrinet switches were used.

Figure 7.6 shows the communication bandwidth performance of PM/Ethernet and TCP/IP on 100BaseT. The maximum bandwidth performance of PM/Ethernet is 11.9MB/s, that of TCP/IP is 11.1MB/s, and these are almost equal to the entire 100BaseT physical bandwidth. The TCP/IP bandwidth performance is higher than that of PM/Ethernet in the case where message length is equal to or less than 256 bytes, because TCP/IP transmits more than one message in one TCP/IP packet[SHT⁺99a],

Figure 7.7 shows PM/Myrinet and PM/Ethernet on Gigabit Ethernet, and communica-

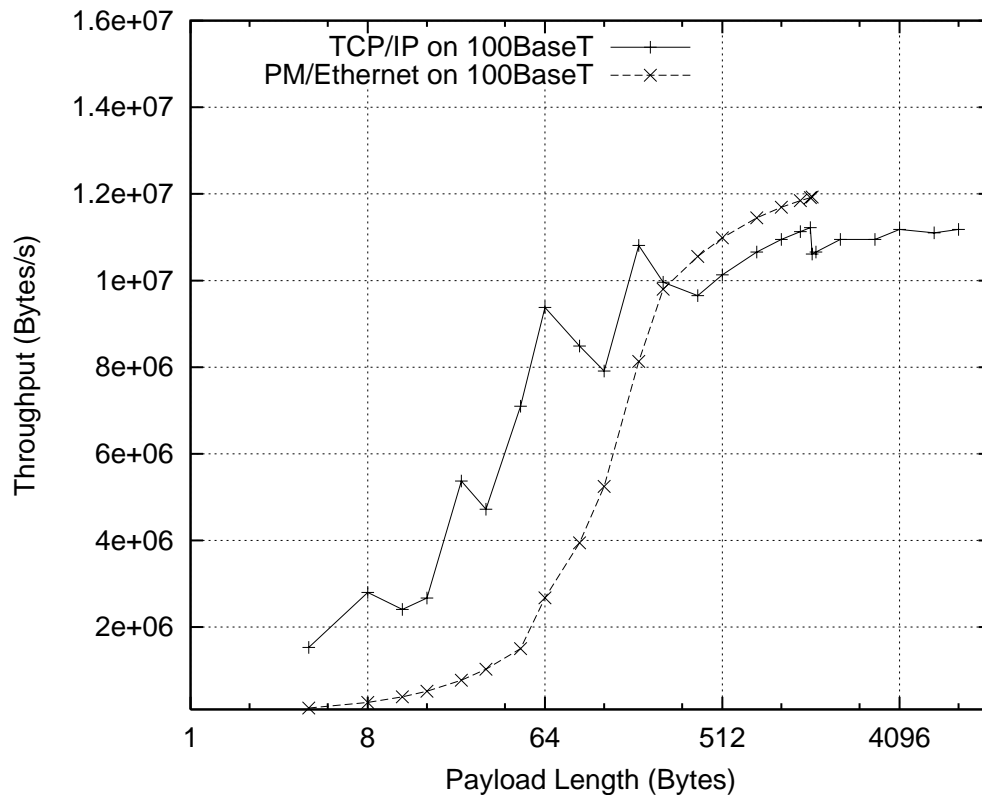


Figure 7.6: Communication Bandwidth on PMv2 (100BaseT)

tion bandwidth performance of TCP/IP on Gigabit Ethernet. The maximum bandwidth performance of PM/Ethernet was 78.4MB/s, TCP/IP was 38.8MB/s and PM/Myrinet was 116.1MB/s.

The bandwidth performance of PM/Ethernet is 2.0 times higher than that of TCP/IP. The TCP/IP bandwidth performance is higher than that of PM/Ethernet in the case where message length is equal to or less than 256 bytes, because TCP/IP transmits more than one message in one TCP/IP packet[SHT⁺99a].

Figure 7.8 shows the communication round trip time of PM/Ethernet and TCP/IP on 100BaseT, Gigabit Ethernet, and PM/Myrinet. In addition, Table 7.5 shows the communication round trip time of a four byte message on PM/Ethernet, PM/Myrinet and TCP/IP. The round trip time of PM/Ethernet on 100BaseT was 7.2 % faster than that of TCP/IP on Gigabit Ethernet.

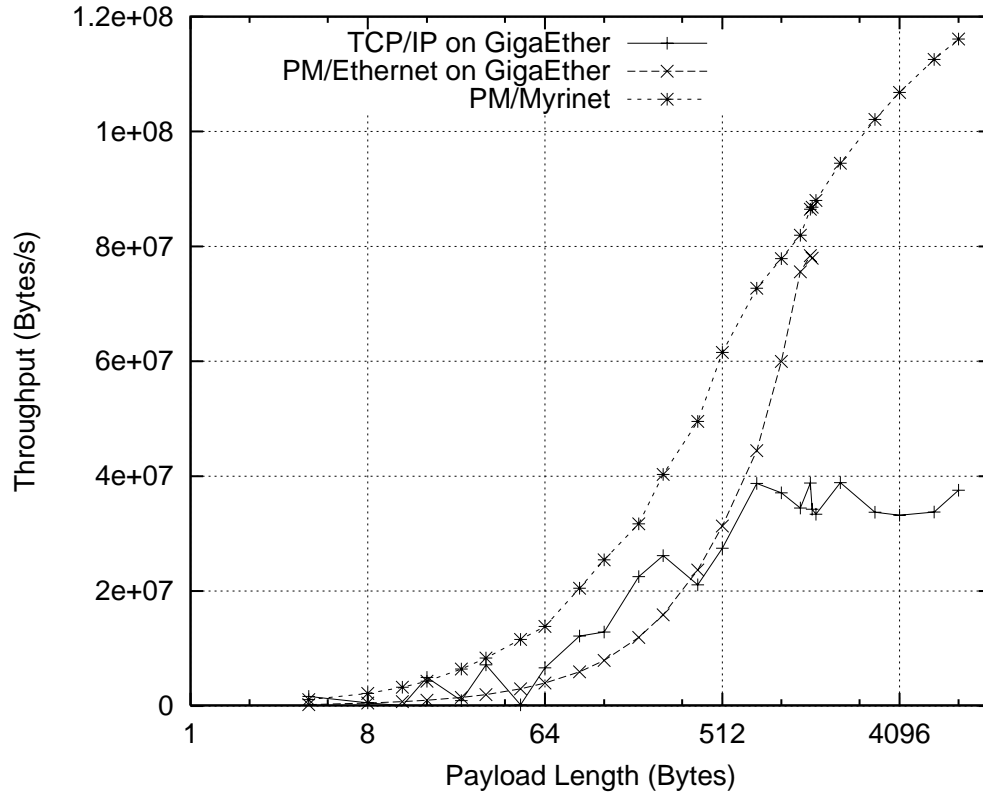


Figure 7.7: Communication Bandwidth on PMv2 (Gigabit Ethernet, Myrinet)

Table 7.5: Communication Round Trip Time on PMv2

Network	RTT
PM/Myrinet	16.4 μs
PM/Ethernet(Gigabit Ethernet)	69.4 μs
TCP/IP(Gigabit Ethernet)	125.0 μs
PM/Ethernet(100BaseT)	115.2 μs
TCP/IP(100BaseT)	178.4 μs

7.4.2 PM/Shmem Round Trip Time and PM/Composite Overhead

This section describes PM/Shmem round trip time and PM/Composite overhead. The overhead of PM/Composite is the same even if a PM device is used, so the PM/Composite overhead using a PM/Shmem device, which is the fastest PM device, is measured.

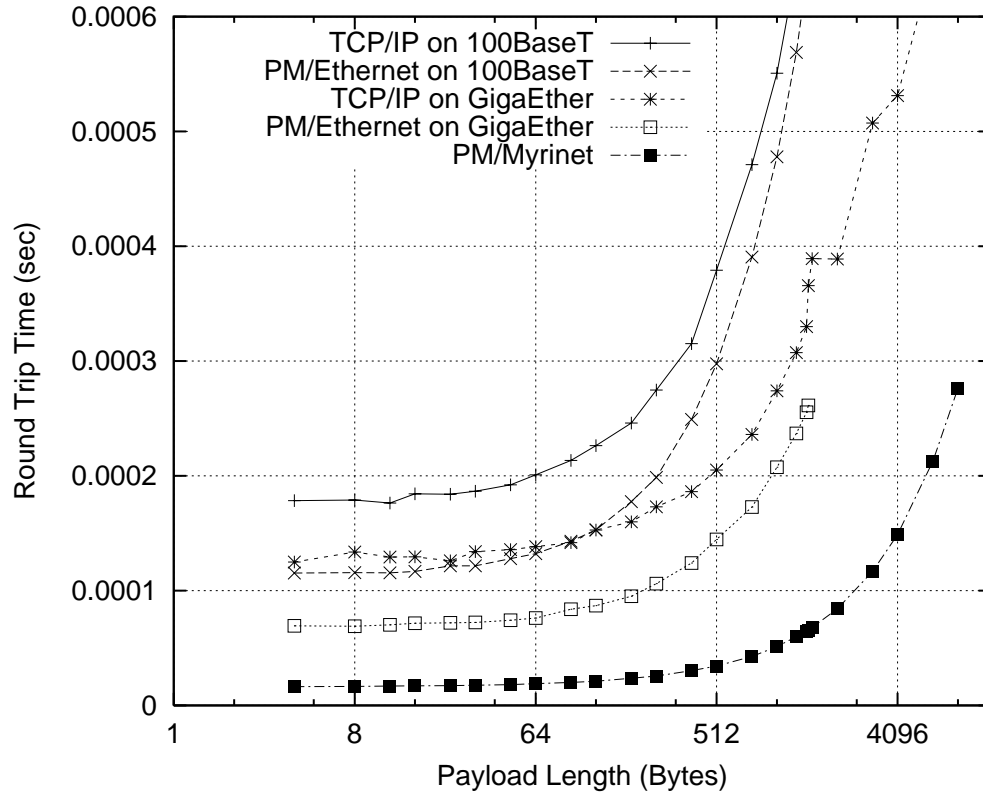


Figure 7.8: Communication Round Trip Time on PMv2

Table 7.6 shows the PM level round trip time (RTT) using PM/Shmem with and without *PM/Composite*. The round trip time of PM/Shmem without *PM/Composite* was $2.75 \mu s$, and that with *PM/Composite* increased by $0.1 \mu s$. Therefore the one way overhead of *PM/Composite* was $0.05 \mu s$ (3.6%). This result shows that the overhead of *PM/Composite* is not crucial even on PM/Shmem.

Table 7.6: Communication Round Trip Time on PM/Shmem

RTT	Without Composite	With Composite
PM/Shmem	$2.75 \mu s$	$2.85 \mu s$

7.4.3 Basic MPI Communication Performance on PM/Myrinet, PM/Ethernet

In this section, the MPI level bandwidth and round trip time are compared with that of TCP/IP. MPICH/SCore for PMv2 and MPI/LAM[LAM] for TCP/IP are used.

Figure 7.9 shows the MPI communication bandwidth performance of PM/Ethernet and TCP/IP on 100BaseT. The maximum bandwidth performance of PM/Ethernet was 11.7M-B/s, that of TCP/IP was 11.MB/s. There was performance degradation of PM/Ethernet in the case where message length is between 768 Bytes and 2048 Bytes, since a method of message transfer was changed from Short-Message-Transfer to Eager-Protocol-Transfer when the message size was longer then 1024 in MPICH/SCore. The Short-Message-Transfer includes control data and transmit data in one message, however the Eager-Protocol-Transfer requires a control message and data messages.

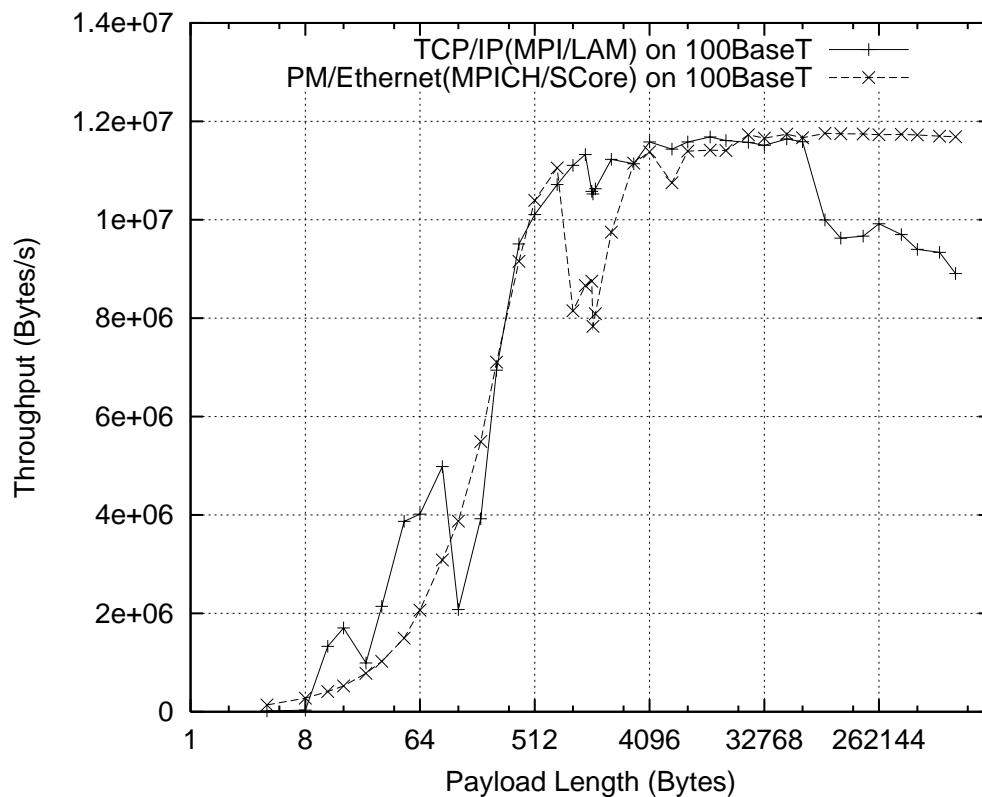


Figure 7.9: MPI Communication Bandwidth on PMv2 (100BaseT)

Figure 7.10 shows the MPI communication bandwidth performance of PM/Ethernet and TCP/IP on Gigabit Ethernet. The maximum bandwidth performance of PM/Myrinet was 67.5, PM/Ethernet was 49.0MB/s, TCP/IP was 36.MB/s. The result of PM/Ethernet is 27 % faster than that of TCP/IP.

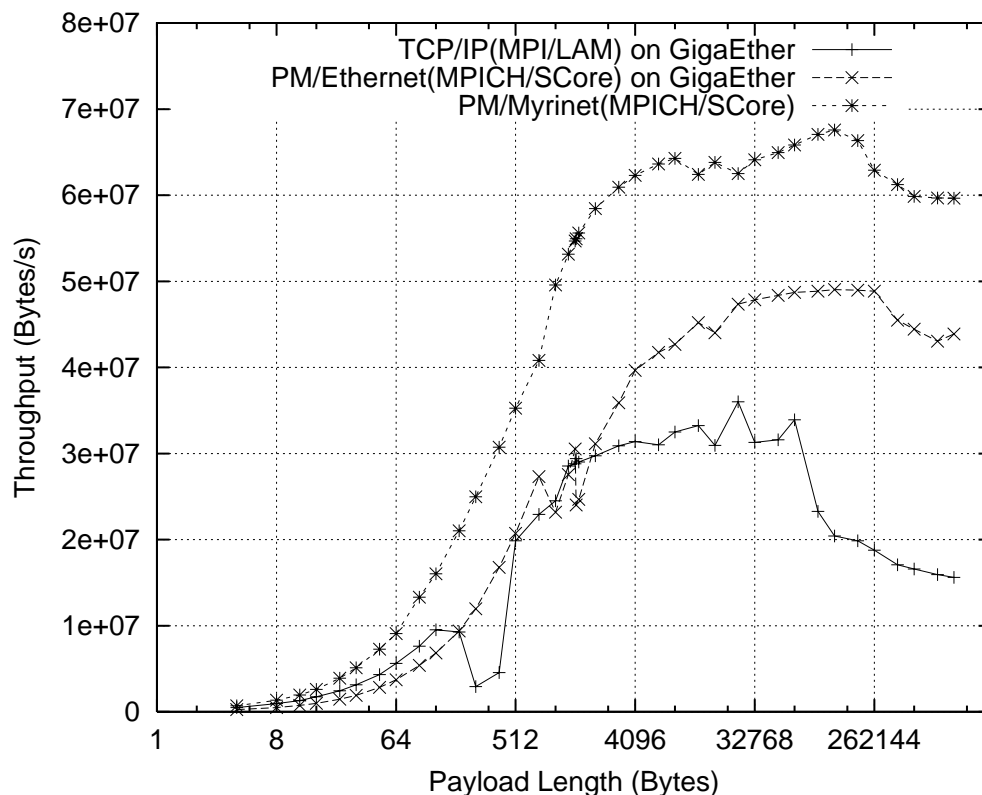


Figure 7.10: MPI Communication Bandwidth on PMv2 (Gigabit Ethernet, Myrinet)

Figure 7.11 shows the MPI communication round trip time of PM/Ethernet and TCP/IP on 100BaseT, Gigabit Ethernet, and PM/Myrinet. In addition, Table 7.7 shows the MPI communication round trip time of a four byte message on PM/Ethernet, PM/Myrinet and TCP/IP. The round trip time of PM/Ethernet on 100BaseT was 24.7 μ s faster than that of TCP/IP on Gigabit Ethernet.

7.4.4 NAS Parallel Benchmark Results

For comparison of true application performance, the NAS parallel benchmarks (Appendix C), version 2.3 class A, were used. The IS (integer sort), CG (Conjugate Gradient), LU

Table 7.7: MPI Communication Round Trip Time

Network		RTT
PM/Myrinet	(MPICH/SCore)	26.2 μs
PM/Ethernet	(MPICH/SCore,Gigabit Ethernet)	90.1 μs
TCP/IP	(MPI/LAM,Gigabit Ethernet)	169.1 μs
PM/Ethernet	(MPICH/SCore,100BaseT)	144.4 μs
TCP/IP	(MPI/LAM,100BaseT)	217.2 μs

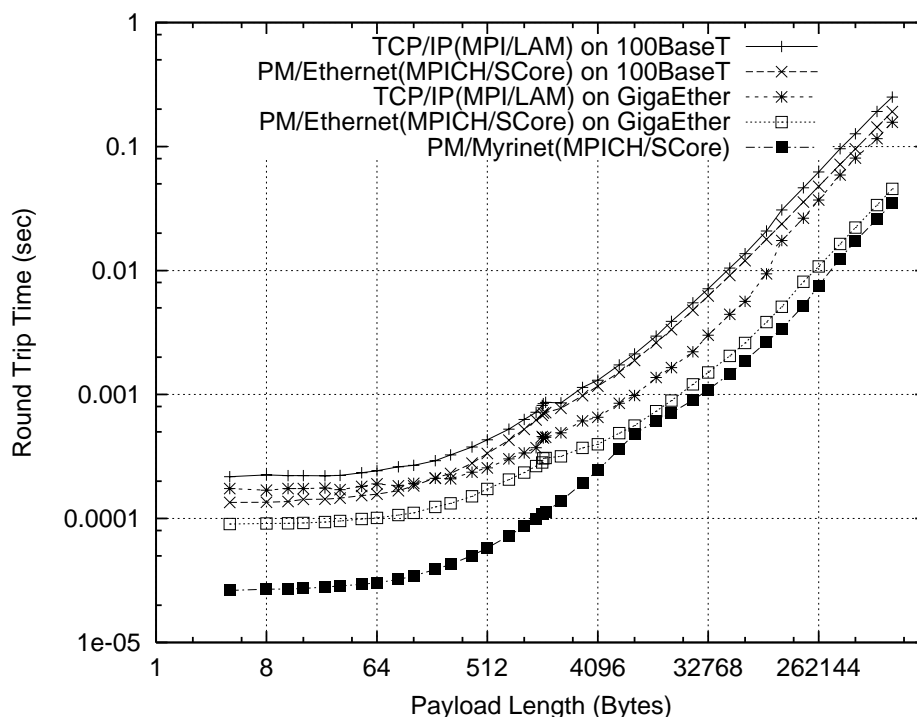


Figure 7.11: MPI Communication Round Trip Time on PMv2

(LU Decomposition) and BT(Block Tridiagonal Solver) in the NAS parallel benchmarks are shown in Figures 7.12, 7.13, 7.14, and 7.15 respectively. All of the NAS parallel benchmarks are shown in Appendix D. In the legend of these figures, GigaEther means Gigabit Ethernet. MPI/LAM[LAM] for TCP/IP is used for as a communication facility which does not use any high-performance communication techniques.

Applications which are Sensitive to Communication Performance:

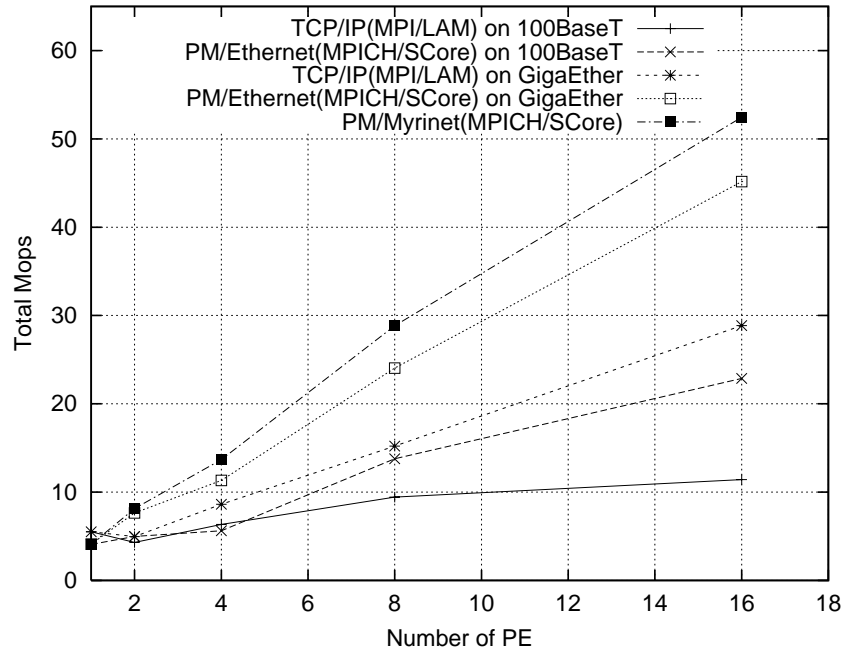


Figure 7.12: An Application which is Sensitive to Communication Performance: IS CLASS A

From the results shown in Figure 7.12, PM/Ethernet is 2.1 times faster, and PM/Myrinet is 2.5 times faster, compared with the performance of TCP/IP. As discussed in section 4.5.1, the dominant message size of IS Class A on 16 PE is in 64–256KB[TSH⁺00]. From the results shown in Figure. 7.10 and 7.11, MPI bandwidth of PM/Ethernet is about 2.5 times higher than that of TCP/IP. MPI RTT of PM/Ethernet is about half of that of TCP/IP around the message size, and MPI bandwidth of PM/Myrinet is about 3.0 times faster than that of TCP/IP and MPI RTT of PM/Myrinet is about one thirds of that of TCP/IP, so, the result of IS is related to the results of MPI communication bandwidth and latency.

These results show that IS performance on Gigabit Ethernet is comparable in performance to that of Myrinet on SCore Cluster I, and about 2 times faster than TCP/IP as a communication without any high-performance techniques.

As for the results of CG shown in Figure 7.13, the performance of PM/Ethernet is better than TCP/IP on Gigabit Ethernet, but, on 100BaseT, the performance on PM/Ethernet is almost the same as TCP/IP because of a lack of bandwidth.

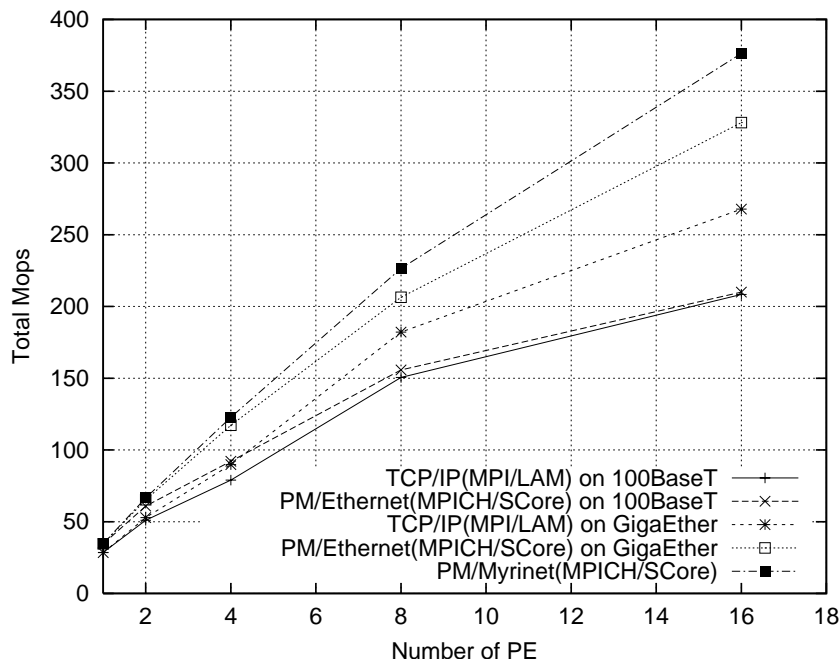


Figure 7.13: An Application which is Sensitive to Communication Performance: CG CLASS A

Applications which are not Sensitive to Communication Performance:

From the results shown in Figures 7.14 and 7.15, there is little performance difference, except for that of TCP/IP on 100BaseT, between the results of LU and BT on all networks.

7.4.5 Application Performance on an SMP Cluster

In this section, the application performance of an SMP cluster is evaluated. The case where two benchmark program processes are executed on one node is measured and compared to the case where one benchmark program process is executed on one node. The results of the IS and LU benchmarks are shown in Figure 7.16 and Figure 7.17.

These figures show the results of executing a benchmark program with two processes on one node. For example, 32 PEs of SMP is a result with 16 nodes \times two processes.

From the results shown in Figure 7.16, a performance improvement is not achieved using PM/Ethernet on 100BaseT, a performance degradation of 33 % is found in PM/Ethernet on Gigabit Ethernet, and degradation of 29 % in PM/Myrinet on 16 PEs. This is because

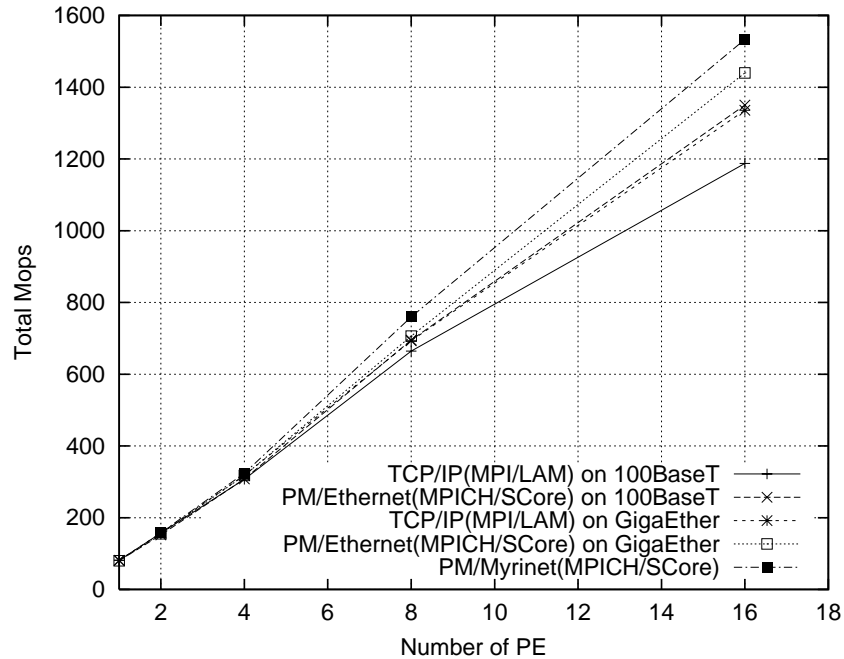


Figure 7.14: An Application which is not Sensitive to Communication Performance: LU CLASS A

of the increase of delay due to the sharing of one network by more than one process.

In contrast to the results for IS, the results for LU in Figure 7.17 are a bit more attractive, a performance degradation was only 10 % on 100BaseT, 9 % on Gigabit Ethernet and 7 % on Myrinet with 16 PEs.

7.4.6 Summary of Results on SCore Cluster I

- Application performance on Gigabit Ethernet is comparable in performance to that of Myrinet on a 16 node cluster.

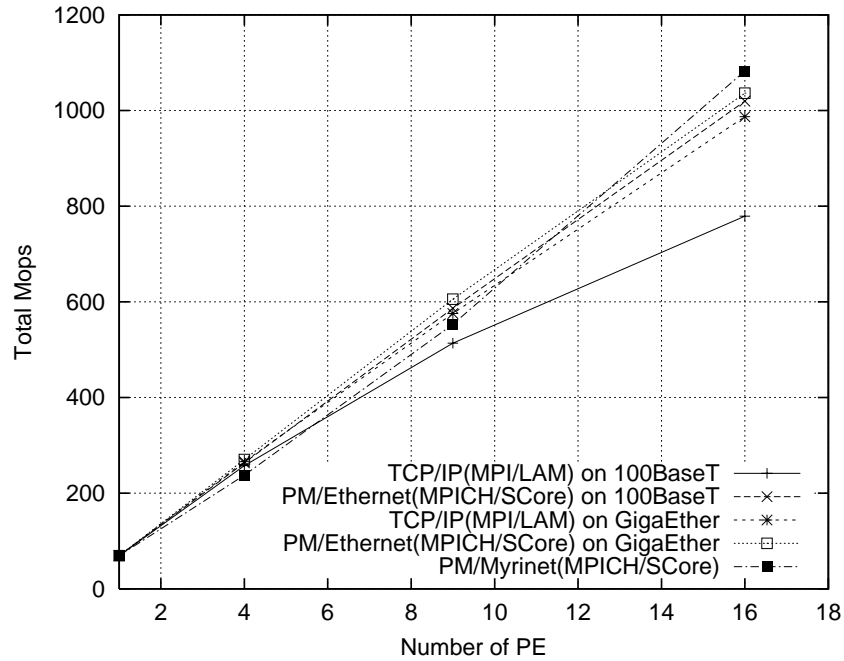


Figure 7.15: An Application which is not Sensitive to Communication Performance: BT CLASS A

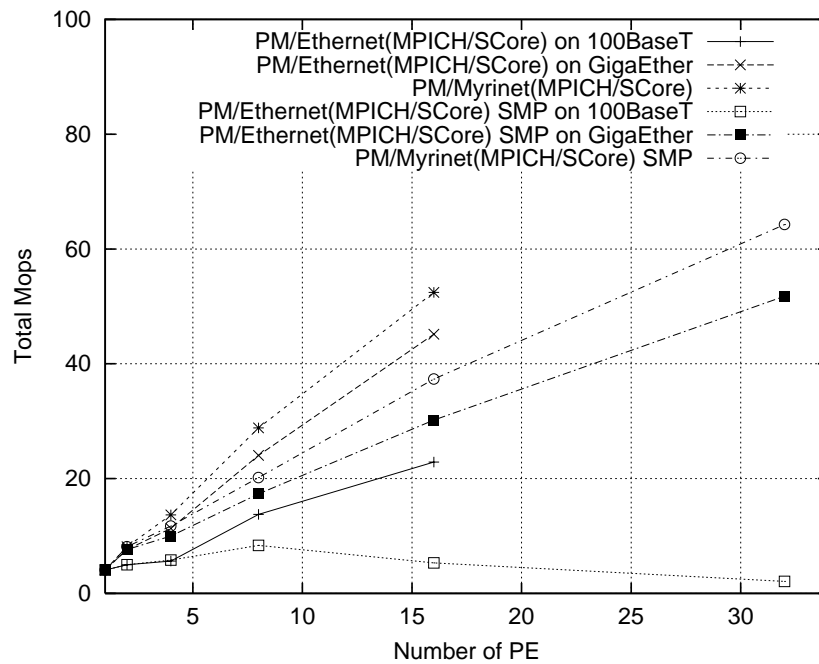


Figure 7.16: An Application which is Sensitive to Communication Performance: IS CLASS A on SMP

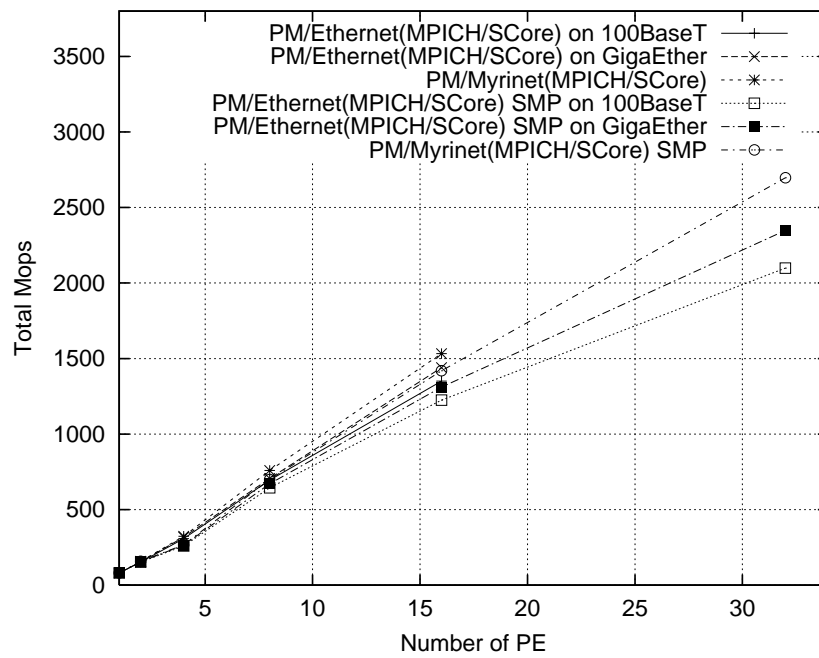


Figure 7.17: An Application which is not Sensitive to Communication Performance: LU CLASS A on SMP

7.5 Performance Evaluation on RWC SCore Cluster II

This section compares application performance of PM/Ethernet using the *Network Trunking* technique to that of PM/Myrinet on SCore Cluster II. The measurement environment is shown in Table 7.8.

Table 7.8: Measurement Environment on RWC SCore Cluster II

Node PC	DUAL Pentium III 800MHz (Serverworks ServerSet III LE chipset, 512MB SDRAM 64 bit 33 MHz PCI bus)
Ethernet NIC	Syskonnect SK9843 (64bit PCI,Gigabit Ethernet) 3 x Intel EEPRO/100(100BaseT)
Ethernet Switch	Summit 7i(Gigabit Ethernet) NEC N468G-N01(100BaseT)
Myrinet	Myricom M2M-PCI64-2 (64 bit PCI) 66 MHz LANai7 processor with 1 MB memory
Myrinet Switch	Myricom M2M-OCT-SW8
OS	Redhat 6.2 Linux (2.2.16 kernel)
SK9843	sk98lin:v1.1.1.1 Date: 2000/06/19 06:44:18
Device Driver	Written by Syskonnect
EEPRO/100	eepro100.c:v1.09j-t 9/29/99
Device Driver	Written by Donald Becker

The Syskonnect Gigabit Ethernet NIC and Myrinet NIC are connected by a 64 bit 33 MHz PCI bus, and three EEPRO100 NICs are connected by a 32 bit 33 MHz PCI bus.

7.5.1 Basic PM Communication Performance

The bandwidth and round trip time on PM/Myrinet, PM/Ethernet on Gigabit Ethernet, and PM/Ethernet using the *Network Trunking* technique are compared in this section.

Figure 7.18 shows the results of the PM level round trip time on RWC SCore Cluster I. From the results shown in Figure 7.18, the round trip time of PM/Myrinet is $13.3 \mu s$ and that of PM/Ethernet on Gigabit Ethernet is $43.3 \mu s$. The results of PM/Ethernet with the *Network Trunking* technique are not greatly different due to the difference of the number of NICs involved.

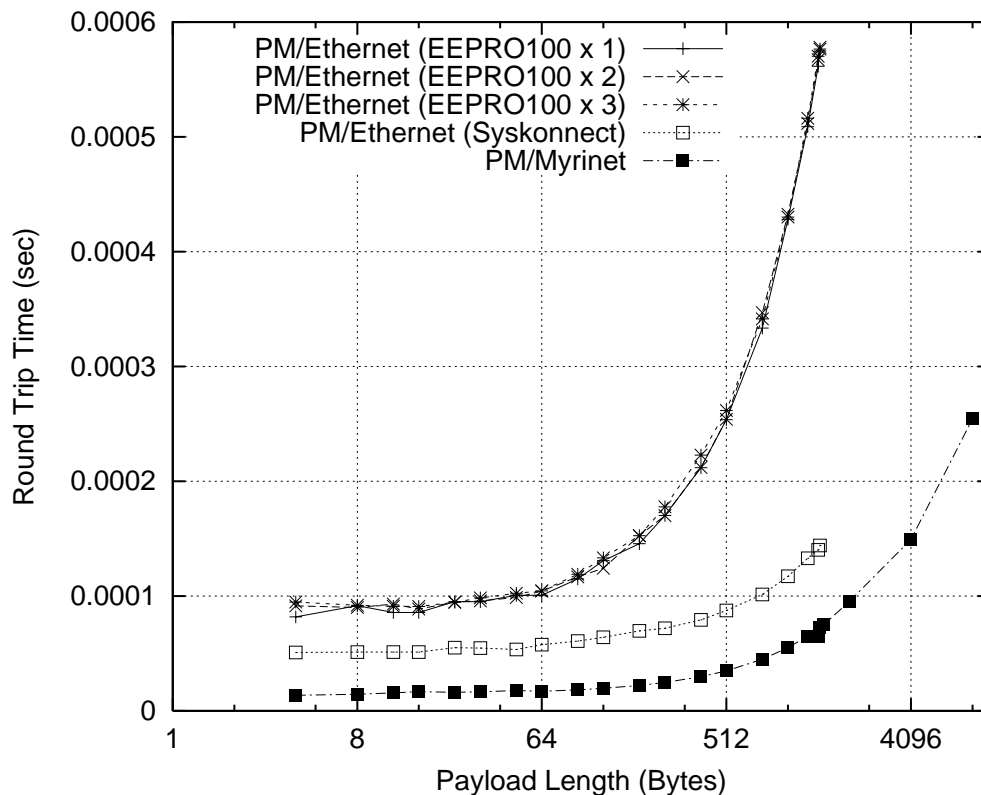


Figure 7.18: Communication Round Trip Time on SCore Cluster II

Figure 7.19 shows the results of the PM level bandwidth on RWC SCore Cluster I. From the results shown in Figure 7.19, PM/Myrinet achieves 146.5 MB/s bandwidth with a 8192 byte message, and PM/Ethernet on Gigabit Ethernet achieves 103.4 MB/s with a 1468 byte message because Syskonnnect and Myrinet NICs are connected on a 64 bit PCI bus in addition to other improvements in their hardware.

PM/Ethernet on 100BaseT with the *Network Trunking* technique also achieves good scalability, until three NICs are used.

7.5.2 Application Performance on RWC SCore Cluster II

Figure 7.20 shows the results of NAS parallel benchmarks (Appendix C), class B, on 16 nodes.

From the results shown in Figure 7.20, the results of the BT, LU and SP benchmarks do not greatly differ due to the difference of network. This shows that the communica-

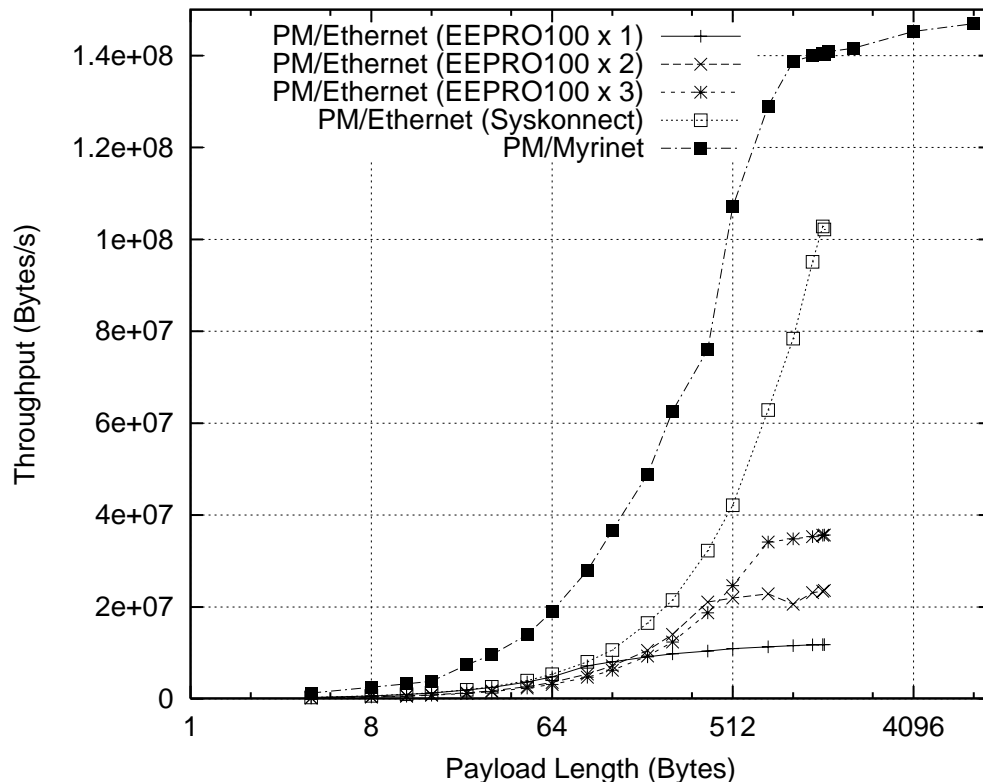


Figure 7.19: Communication Bandwidth on SCore Cluster II

tion performance needed in BT, LU and SP benchmarks are enough to that of 100BaseT Ethernet on a 16 node cluster.

In contrast to these results, the results of the CG, FT and IS benchmarks are sensitive to the difference of network. PM/Ethernet with the *Network Trunking* technique on FT and IS achieved dramatic results, however the results on CG did not increase as dramatically. This is because CG requires high communication bandwidth for neighbor communication, and the results reflect this. FT and IS use “all-to-all” communication frequently, which transfers many messages to the other nodes, so, in the case of one 100BaseT network, many re-transmission occurred and decreased the performance.

7.5.3 Summary of Results on SCore Cluster II

- Application performance on Gigabit Ethernet is comparable in performance to that of Myrinet on a 16 node cluster.

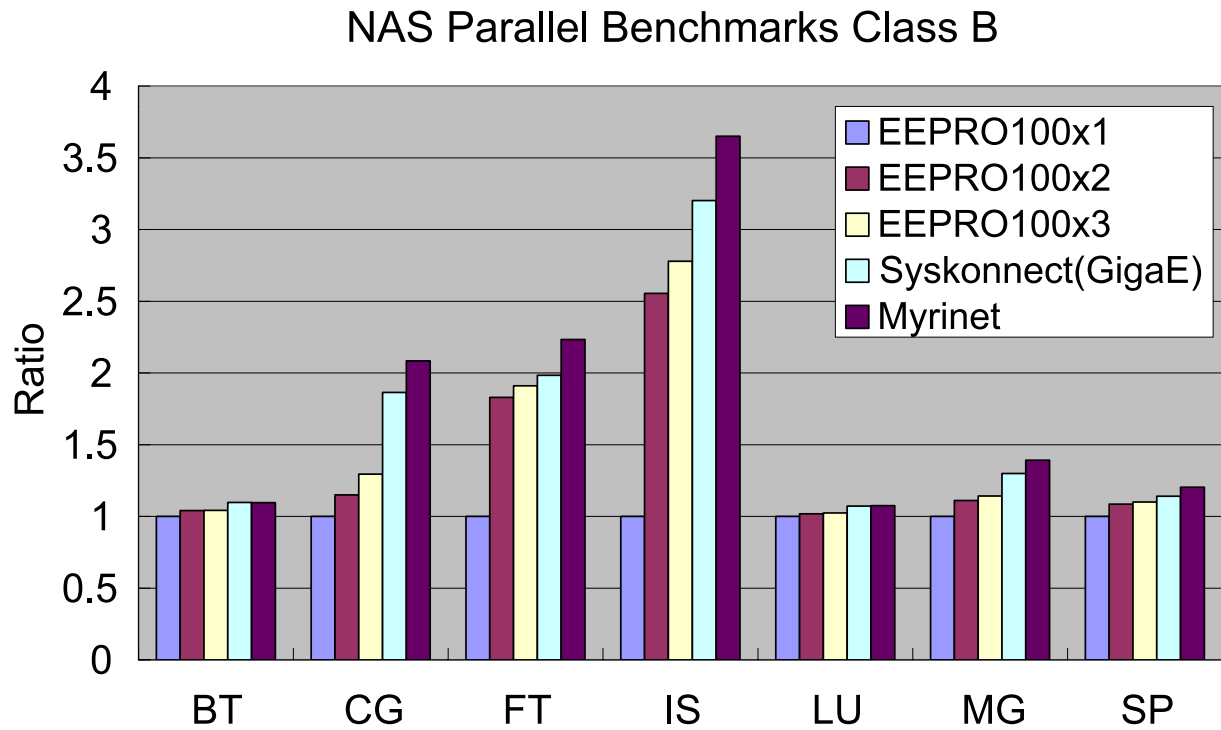


Figure 7.20: NAS Parallel Benchmarks, Class B, 16 nodes, on SCore Cluster II

- The *Network Trunking* technique increases performance on all benchmark programs, the performance improvement of IS and FT is especially remarkable.

7.6 Performance Evaluation on RWC PC Cluster II

This section evaluates the scalability of NAS parallel benchmarks (Appendix C) of P-M/Ethernet on Fast Ethernet (100BaseT) to PM/Myrinet on RWC PC Cluster II (128 node cluster). The measurement environment is shown in Table 7.9. All of the NAS parallel benchmarks are shown in Appendix E.

Table 7.9: Measurement Environment on RWC PC Cluster II

Node PC	Pentium PRO 200MHz (440FX chipset, 256MB EDO-DRAM 32 bit 33MHz PCI bus)
Ethernet NIC	Digital 21140A (100BaseT)
Ethernet Switch	4 x 3Com SuperStackII 3900(100BaseT)
Myrinet	Myricom M2M-PCI32 33 MHz LANai4 processor with 1 MB memory
Myrinet Switch	Myricom M2M-OCT-SW8
Host OS	Redhat 6.1 Linux (2.2.16 kernel)
21140A Device Driver	tulip.c:v0.91g-ppc 7/16/99 Written by Donald Becker

Applications which are not Sensitive to Communication Performance:

Figure 7.21 shows the EP Class B results. As shown in given by Figure 7.21, the performance of EP does not almost depend on the network, because it does not use the network except for start and finish synchronization.

Figure 7.22 shows the BT Class B results. As shown in given by Figure 7.22, BT on Myrinet, PM/Ethernet and TCP/IP, achieve good performance scalability even on a 128 node cluster.

Figure 7.23 shows the NPB LU Class B results. From Figure 7.23, LU on Myrinet, PM/Ethernet and TCP/IP achieve good performance scalability even on a 128 node cluster. However the performance difference between 100BaseT and Myrinet becomes bigger from 32-33 PEs.

Applications which are Sensitive to Communication Performance:

Figure 7.24 shows the NPB CG Class B results. From Figure 7.24, the performance of

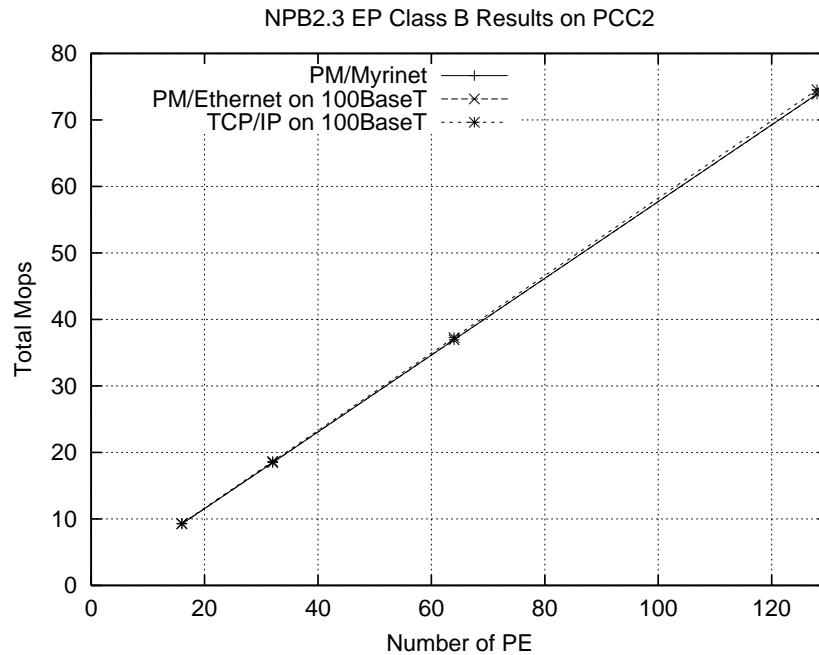


Figure 7.21: An Application which does not Sensitive to Communication Performance: EP CLASS B

CG on Myrinet is about 2 times faster than that of PM/Ethernet and TCP/IP on 128 PEs, because of the lack of communication bandwidth.

Figure 7.25 shows the NPB IS Class B results. From Figure 7.25, the performance of IS on Myrinet is about 2 times faster than that of PM/Ethernet. The IS performance on TCP/IP on 128 PEs is worse than that on 16 PEs.

7.6.1 Summary of Results on PC Cluster II

- The performance of IS on Myrinet is about 2 times faster than that of PM/Ethernet.

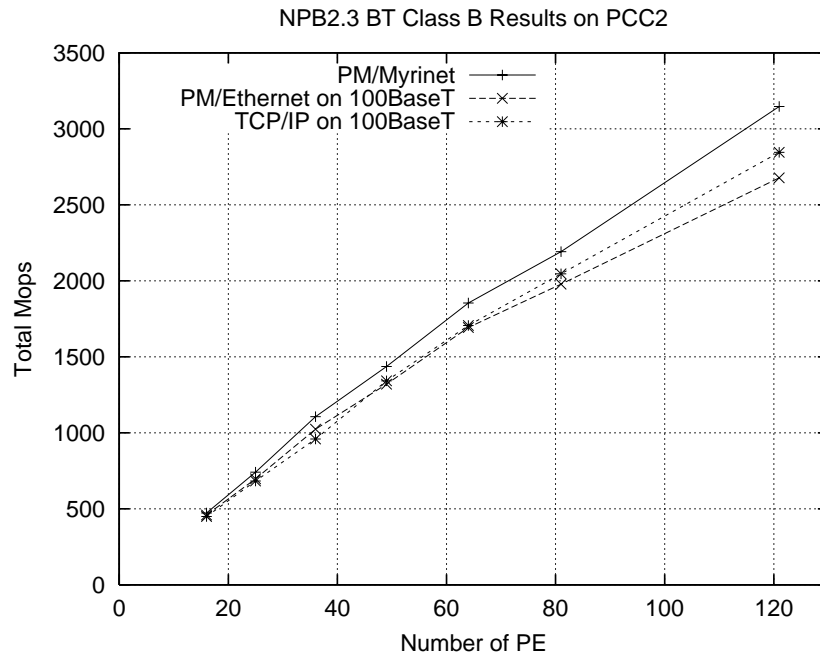


Figure 7.22: An Application which is not Sensitive to Communication Performance: BT CLASS B

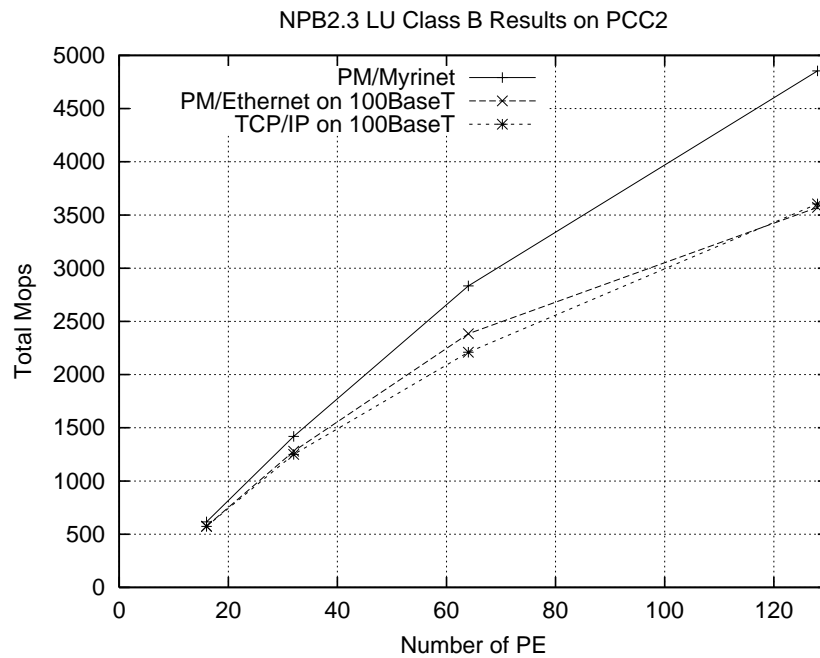


Figure 7.23: An Application which is not Sensitive to Communication Performance: LU CLASS B

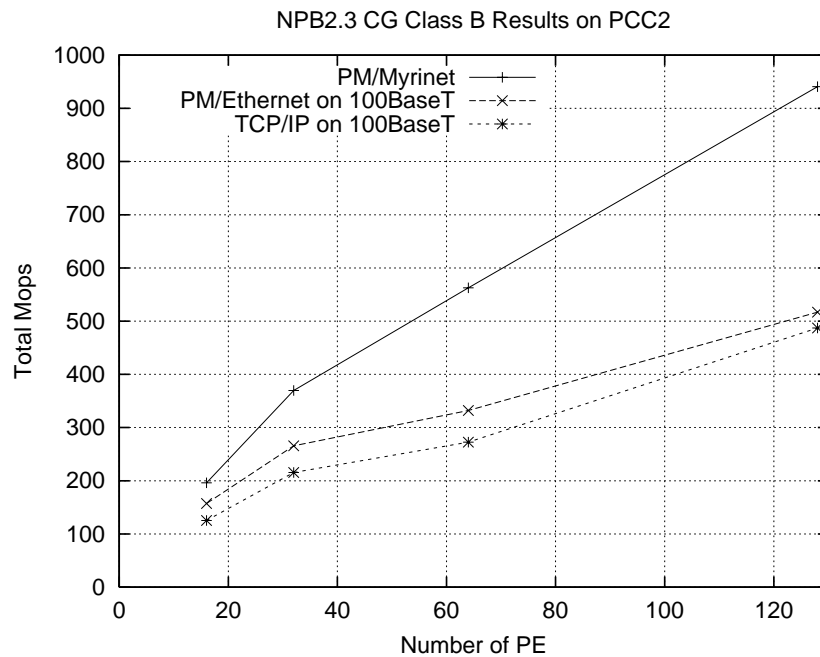


Figure 7.24: An Application which is Sensitive to Communication Bandwidth: CG CLASS B

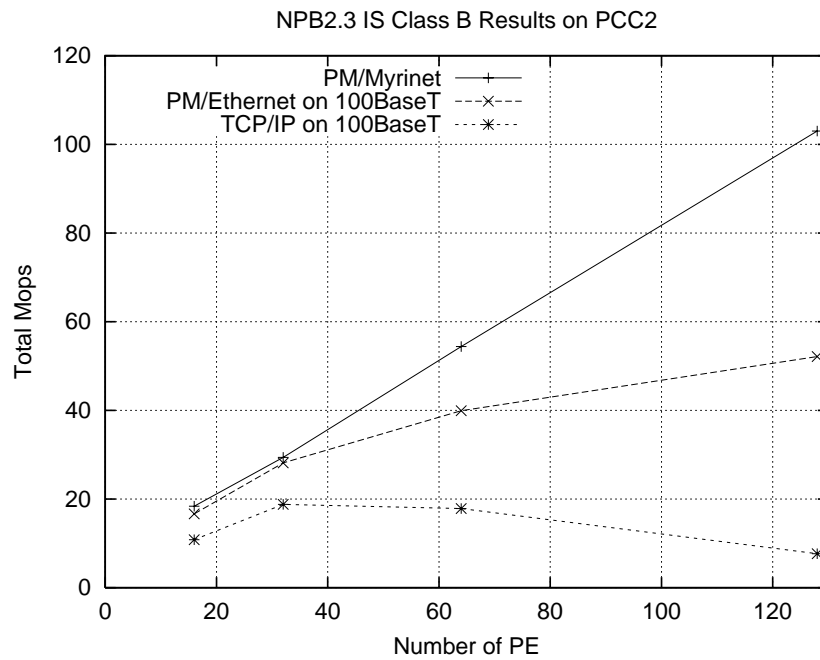


Figure 7.25: An Application which is Sensitive to Communication Latency: IS CLASS B

7.7 Conclusions of This Chapter

This chapter compares application performance of PM/Ethernet using the *Network Trunking* technique to PM/Myrinet on RWC clusters. In order to compare more precisely, a high performance communication facility, PMv2, has been designed and implemented.

A difference in application performance is evaluated using the NAS parallel benchmarks on RWC SCore Cluster I, RWC SCore Cluster II and RWC PC Cluster II, and found that:

- Application performance on Gigabit Ethernet is comparable to that on Myrinet on a 16 node cluster.
- The *Network Trunking* technique improves performance on all benchmark programs.
- The performance difference between PM/Myrinet and PM/Ethernet on Fast Ethernet is about 2 times at most.

These results show that a practical high-performance cluster system can be built using a commodity network comparable in performance to a dedicated cluster system network.

Chapter 8

Conclusion

This thesis has been done in order to achieve the following purposes: proposing implementation designs which ensure the maximum communication performance on cluster systems using Gigabit Ethernet; and showing that communication using Gigabit Ethernet can achieve comparable performance of application programs to dedicated cluster network communication.

To achieve the purposes, the following designs are proposed:

- Hardware dependent designs
- Hardware independent designs using existing NICs
- Software techniques using multiple NICs

For hardware dependent designs, the following techniques are proposed: a reliable lightweight communication protocol called "GoBack-N for PM" and also design techniques which minimize information exchange cost based on a cost analysis of TCP/IP.

The GigaE PM communication facility has been implemented using these techniques on the Essential Gigabit Ethernet NIC. The benchmark results show that the communication bandwidth of GigaE PM is 1.7 times faster than that of the existing TCP/IP protocol, and the round trip time of GigaE PM is about one third that of TCP/IP. Therefore, the proposed method is effective in realizing high performance communication on Gigabit Ethernet.

For hardware independent designs, a design method which reduces reliable protocol processing overhead is proposed. The method is based on the results of a cost analysis of the

existing TCP/IP processing overhead.

According to the analysis, the interrupt overhead caused performance degradation in addition to the TCP/IP protocol processing overhead. In order to eliminate the interrupt overhead, a communication protocol processing method has been proposed. This method does not use a software interrupt as existing protocols do. The *Interrupt Reaping* technique has also been proposed. The use of a hardware interrupt is eliminated without modification of existing device drivers.

The PM/Ethernet communication facility has been implemented using these techniques on the Packet Engines G-NIC II Gigabit Ethernet NIC. The performance benchmark results show that the communication bandwidth of PM/Ethernet is 1.6 times faster than that of the existing TCP/IP protocol, and the round trip time of PM/Ethernet is about one third that of TCP/IP. These results show that the proposed method eliminates two thirds of the TCP/IP processing cost.

As an another hardware independent technique, the *Network Trunking* technique has been developed. It improves communication bandwidth using multiple NICs. The communication performance benchmark results show that the bandwidth performance of PM/Ethernet with four digital 100 Base/T NICs is 3.6 times faster than that with one 100 Base/T NIC.

To show that communication using Gigabit Ethernet achieves comparable performance of application programs, a communication facility which supports both a dedicated cluster network and a commodity network has been developed. As a result of the evaluation using the NAS parallel benchmarks, the performance of the IS benchmark on PM/Ethernet using Gigabit Ethernet achieves comparable performance to that on PM/Myrinet on a 16 node cluster. This result shows that practical high-performance cluster systems can be built using a commodity network.

Using the method proposed above, it is verified that a cluster system using a commodity network can achieve comparable performance to that of a cluster system using a dedicated cluster network.

Bibliography

- [Aki99] Yutaka Akiyama. Bioinformatics. In *IP SJ Magazine*, volume 40(11), pages 1136–1138. the Information Processing Society of Japan, 1999.
- [AMB] AMBER information:
<http://www.amber.ucsf.edu/amber/amber.html>.
- [BBVvE97] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the Third International Symposium on High Performance Computer Architecture (HPCA)*, February 1997.
- [BEO] The Beowulf Project:
<http://www.beowulf.org/>.
- [BVI] Berkeley VIA Project:
<http://www.millennium.berkeley.edu/via.php3>.
- [CC00] G. Chiola and G. Ciaccio. Efficient parallel processing on low-cost clusters with GAMMA active ports. In *Parallel Computing*, number 2-3, pages 333–354. North Holland, February 2000.
- [CMC97] B. N. Chun, A. M. Mainwaring, and D. E. Culler. Virtual Network Transport Protocols for Myrinet. In *Hot Interconnect'97*, Aug 1997.
- [CPL] The Computational Plant project:
<http://www.cs.sandia.gov/cplant>.

- [DBC⁺97] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient Support for Reliable, Connection-Oriented Communication. In *Hot Interconnect'97*, August 1997.
- [DBLP97] Cezary Dubnicki, Angelos Bilas, Kai Li, and James Philbin. Design and Implementation of Virtual Memory-Mapped Communication on Myrinet. In *the IEEE 11th International Parallel Processing Symposium*, April 1997.
- [DOL] Dolphin Interconnect:
<http://208.179.47.35/index.html>.
- [ETH] Development of the Earth Simulator:
http://www.gaia.jaeri.go.jp/public/e_publicconts.html.
- [FO00] Paul A. Farrell and Hong Ong. Communication Performance over a Gigabit Ethernet Network. In *19th IEEE International Performance, Computing and Communications Conference (IPCCC 2000)*, pages 181–189, February 2000.
- [GM] The GM API:
http://www.myri.com/GM/doc/gm_toc.html.
- [GPT99] P. Geoffray, L. Prylli, and B. Tourancheau. BIP-SMP: High Performance Message Passing over a Cluster of Commodity SMPs. In *Super computing 99*, Portland, USA, November 1999.
- [HAM] Packet Engines Hamachi Performance:
<http://www.nscl.msu.edu/~kasten/perf/hamachi/>.
- [HCP] Chronology of Personal Computers :
<http://www.islandnet.com/~kpolsson/comphist/>.
- [HIH⁺00] Hiroshi Harada, Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Shinji Sumimoto, and Toshiyuki Takahashi. Dynamic Home Node Reallocation on Software Distributed Shared Memory. In *4Th HPC ASIA 2000*, pages 158–163. IEEE, May 2000.

- [HIK⁺93] Hori, Ishikawa, Konaka, Maeda, and Tomokiyo. Overview of Massively Parallel Operating System Kernel SCORE. In *IPSJ SIG Notes*, 93-OS-61, pages 57–64. Information Processing Society of Japan, August 1993. (In Japanese).
- [HIK⁺95] Atsushi Hori, Yutaka Ishikawa, Hiroki Konaka, Munenori Maeda, and Takashi Tomokiyo. A Scalable Time-Sharing Scheduling for Partitionable, Distributed Memory Parallel Machines. In *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences, Vol. II*, pages 173–182. IEEE Computer Society Press, January 1995.
- [HIN] Intel Museum, Processor Hall of Fame:
http://www.intel.com/intel/museum/25anniv/hof/hof_main.htm.
- [HIN⁺95] Atsushi Hori, Yutaka Ishikawa, Jörg Nolte, Hiroki Konaka, Munenori Maeda, and Takashi Tomokiyo. Time Space Sharing Scheduling: A Simulation Analysis. In S. Haridi, K. Ali, and P. Magnusson, editors, *Euro-Par'95 Parallel Processing*, volume 966 of *Lecture Notes in Computer Science*, pages 623–634. Springer-Verlag, August 1995.
- [HIS⁺93] Hori, Ishikawa, Sakai, Konaka, Maeda, Tomokiyo, Matsuoka, Okamoto, Hirono, and Yokota. Process management in SCORE and hardware support for massively parallel OS. In *Computer System Symposium*, volume 93, pages 59–66. Information Processing Society of Japan, October 1993. (In Japanese).
- [HPV] High Performance Virtual Machines:
<http://www-csag.ucsd.edu/projects/hpvm.html>.
- [HTI⁺96] Atsushi Hori, Hiroshi Tezuka, Yutaka Ishikawa, Noriyuki Soda, Hiroki Konaka, and Munenori Maeda. Implementation of Gang-Scheduling on Workstation Cluster. In D. G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 76–83. Springer-Verlag, April 1996.

- [HTI98] Atsushi Hori, Hiroshi Tezuka, and Yutaka Ishikawa. Highly Efficient Gang Scheduling Implementation. In *SC'98*, November 1998.
- [HTT⁺99] Atsushi Hori, Hiroshi Tezuka, Toshiyuki Takahashi, Shinji Sumimoto, Noriyuki Soda, Hiroshi Harada, and Yutaka Ishikawa. Programming Environment for Clusters – SCORE Cluster System Software –. In *Special Interest Group Report of the Information Processing Society of Japan 99-HPC-77 (SWoPP'99)*, pages 83–88. the Information Processing Society of Japan, August 1999. (In Japanese).
- [HYI⁺95] Atsushi Hori, Takashi Yokota, Yutaka Ishikawa, Shuichi Sakai, Hiroki Konaka, Munenori Maeda, Takashi Tomokiyo, Jörg Nolte, Hiroshi Matsuoka, Kazuaki Okamoto, and Hideo Hirono. Time Space Sharing Scheduling and Architectural Support. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1995.
- [IHK⁺93] Y. Ishikawa, A. Hori, H. Konaka, M. Maeda, and T. Tomokiyo. MPC++: A Parallel Programming Language and Its Parallel Objects Support. In *OOPSLA 1993 Workshop on Efficient Implementation of Concurrent Object-Oriented Languages*, pages j1–j5, September 1993.
- [IHK⁺94] Yutaka Ishikawa, Atsushi Hori, Hiroki Konaka, Maeda Munenori, and Tomokiyo Takashi. Implementation of parallel programming language MPC++. In *JSPP'94*, pages 105–112, 1994.
- [IHS⁺95] Yutaka Ishikawa, Atsushi Hori, Mitsuhisa Sato, Motohiko Matsuda, Jörg Nolte, Hiroshi Tezuka, Hiroki Konaka, Munenori Maeda, and Takashi Tomokiyo. An overview of mpc++ – extended abstract –. In Takayasu Ito and Jr. Robert H. Halstead, editors, *International Workshop PSLs'95*, volume 1068 of *Lecture Notes in Computer Science*, pages 243–249. Springer-Verlag, October 1995.

- [IHT⁺96] Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Motohiko Matsuda, Hiroki Konaka, Munenori Maeda, Takashi Tomokiyo, and Jörg Nolte. MPC++. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming Using C++*, pages 429–464. MIT Press, 1996.
- [Ish96] Yutaka Ishikawa. Multi Thread Template Library – MPC++ Version 2.0 Level 0 Document –. Technical Report TR–96012, RWC, September 1996.
- [ISS96] Yutaka Ishikawa, Mitsuhsa Sato, and Junichi Shimada. Toward A Seamless Computing Environment. Technical Report TR–96013, RWC, September 1996.
- [ITH⁺99] Yutaka Ishikawa, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, Francis O’Carroll, and Hiroshi Harada. RWC PC Cluster II and SCORE Cluster System Software – High Performance Linux Cluster. In *Proceedings of the 5th Annual Linux Expo*, pages 55 – 62, 1999.
- [JR86] Michael B. Jones and Richard F. Rashid. Mach and matchmaker: Kernel and language support for object-oriented distributed systems. In *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA ’86)*, volume SIGPLAN Notices 21, pages 67–77, November 1986.
- [KISB99] Kazuto Kubota, Ken’ich Itakura, Mitsuhsa Sato, and Taiske Boku. Practical Simulation of Large-Scale Parallel Programs and its Performance Analysis of the NAS Parallel Benchmarks. In *Euro-Par’98 Parallel Processing*, pages 244–254, January 1999.
- [LAM] LAM / MPI Parallel Computing:
<http://www.mpi.nd.edu/lam/>.
- [LMC97] S. S. Lumetta, A. M. Mainwaring, and D. E. Culler. Multi-protocol Active messages on a cluster of smp’s. In *Super Computing(SC’97)*, November 1997.
- [MEM] Architecture and Implementation of MEMORY CHANNEL2:
<http://research.compaq.com/wrl/DECarchives/DTJ/DTJP03/DTJP03HM.HTM>.

- [MPIa] The Message Passing Interface (MPI) standard:
<http://www.mpi-forum.org>.
- [MPIb] MPICH-A Portable Implementation of MPI:
<http://www-unix.mcs.anl.gov/mpi/mpich/>.
- [MVI] M-VIA: A High Performance Modular VIA for Linux:
<http://www.nersc.gov/research/FTG/via/>.
- [MW97] T. von Eicken M. Welsh, A. Basu. Incorporating Memory Management into User-Level Network Interfaces . In *Proc. of Hot Interconnects V*, August 1997.
- [MYR] Myricom:
<http://www.myri.com>.
- [N. 95] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su. Myrinet – A Gigabit-per-Second Local-Area Network. *IEEE MICRO*, 15(1):29–36, February 1995.
- [NET] The Public Netperf:
<http://www.netperf.org/>.
- [NOW] The Berkeley Network of Workstations (NOW) project:
<http://now.cs.berkeley.edu/>.
- [NPB] The NAS Parallel Benchmarks (NPB):
<http://www.nas.nasa.gov/Software/NPB/>.
- [OCD⁺88] John K. Ousterhout, Andrew R. Cherenson, Fred Douglass, Michael N. Nelson, and Brent B. Welch. The sprite network operating system. In *IEEE Computer*, volume 21, pages 23–36, 1988.
- [OHT⁺97] Francis O’Carroll, Atsushi Hori, Hiroshi Tezuka, Yutaka Ishikawa, and Mitsuhiro Sato. Performance of MPI on Workstation/PC Clusters using Myrinet. In *Proceedings of Cluster Computing Conference ’97*, March 1997.

- [OTHI98] Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *ICS'98*, pages 243 – 250, July 1998.
- [PCI] PCI technical briefs:
<http://www.intel.com/product/tech-briefs/pcibus.htm>.
- [PD80] D. A. Patterson and D. R. Ditzel. The case for the reduced instruction set computer. In *Computer Architecture News*, volume 8, pages 25–33, October 1980.
- [PIC] PCI Industrial Computer Manufacturers Group:
<http://www.picmg.org/>.
- [PMA] PM 2.1 API:
<http://pdswww.rwcp.or.jp/dist/score/html/reference/man/man3/PM.html>.
- [PT98] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance. In PC-NOW Workshop, held in parallel with IPPS/SPDP98, Orlando, USA, Mar 30 – Apr 3 1998.
- [PVM] Parallel Virtual Machine:
http://www.epm.ornl.gov/pvm/pvm_home.html.
- [RR81] Richard F. Rashid and George G. Robertson. Accent: A communication oriented network operating system kernel. In *the Eighth ACM Symposium on Operating System Principles*, volume Operating System Review 15, pages 64–75, December 1981.
- [RTO] Overview of Recent Supercomputers:
<http://www.top500.org/ORSC/>.
- [RWC] Clustering Technologies at RWCP:
<http://pdswww.rwcp.or.jp/clusters/home.html>.

- [SCI] The Local Area Memory Port, Local Area MultiProcessor, Scalable Coherent Interface, and Serial Express Users, Developers, and Manufacturers Association:
<http://www.scizzl.com/>.
- [SCL] Gigabit Ethernet and Low-Cost Supercomputing:
<http://www.scl.ameslab.gov/Publications/Gigabit/tr5126.html>.
- [Sco95] Scott Pakin, Mario Lauria and Andrew Chein. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing '95, San Diego, California, 1995*.
- [SET] IEEE 802.3 CSMA/CD (ETHERNET):
<http://grouper.ieee.org/groups/802/3/>.
- [SHR] The SHRIMP project:
<http://www.cs.princeton.edu/shrimp/>.
- [SHT⁺99a] Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa. GigaE PM II: Design of High Performance Communication Library using Gigabit Ethernet. In *Special Interest Group Report of the Information Processing Society of Japan 99-ARC-134 (SWoPP'99)*, pages 61–66. the Information Processing Society of Japan, August 1999. (In Japanese).
- [SHT⁺99b] Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa. Performance evaluation of gigabit ethernet nic. In *Special Interest Group Report of Information Processing Society of Japan 99-HPC-75 (HOKKE'99)*, pages 49–54. the Information Processing Society of Japan, March 1999. (In Japanese).
- [SHT⁺00a] Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa. High Performance Communication for Cluster Systems using an Existing Operating System Framework. In *Transactions*

- of the Information Processing Society of Japan, Vol. 41, Number 6*, pages 1688–1696. the Information Processing Society of Japan, June 2000. (in Japanese).
- [SHT⁺00b] Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa. The Design and Evaluation of High Performance Communication Facility: PM2. In *Transactions of the Information Processing Society of Japan, Vol. 41 No.SIG 5 (HPS 1)*, pages 80–90, August 2000.
- [SIP] Peripheral Component Interconnect(PCI) Special Interest Group (PCI SIG): <http://www.pcisig.com/>.
- [SIS] ISA BUS OVERVIEW:
http://quatech.salcomm.com/Application_Objects/FAQs/comm-over-isa.htm.
- [SPC] PCI BUS OVERVIEW :
http://quatech.salcomm.com/Application_Objects/FAQs/comm-over-pci.htm.
- [SPE] The Standard Performance Evaluation Corp (SPEC):
<http://www.spec.org/>.
- [Ste] W. Richard Stevens. In *UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI*. Prentice Hall.
- [T. 92] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauer. Active messages: a Mechanism for Integrated Communication and Computation. In *In Proc. of the 19th ISCA*, pages 256–266, May 1992.
- [T. 94] T. von Eicken, V. Avula, A. Basu and V. Buch. Low-Latency Communication over ATM Networks using Active Messages. In *In Proceedings of Hot Interconnects II, 1994 Palo Alto*, August 1994.

- [T. 95] T. Sterling, D. Savarese, D. J. Becker, B. Fryxell, K. Olson. Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation. In *Proceedings of the Fourth IEEE Symposim on High Performance Distributed Computing(HPDC-95)*, August 1995.
- [THI96] Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. Design and Implementation of PM: A Communication Library for Workstation Cluster. In *JSPP'96*, pages 41–48. the Information Processing Society of Japan, June 1996. (In Japanese).
- [THIS97] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhsa Sato. PM: An Operating System Coordinated High Performance Communication Library. In Peter Sloot Bob Hertzberger, editor, *High-Performance Computing and Networking*, volume 1225 of *Lecture Notes in Computer Science*, pages 708–717. Springer-Verlag, April 1997.
- [TISY] Toshiyuki Takahashi, Yutaka Ishikawa, Mitsuhsa Sato, and Akinori Yonezawa. Class specific optimization environment using compile-time metalevel architecture. In *ISCOPE'97*.
- [TLC85] Marvin Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the v-system. In *the Tenth ACM Symposium on Operating System Principles*, volume Operating System Review 19, pages 2–12, December 1985.
- [TOHI98] Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *IPPS/SPDP'98*, pages 308–314. IEEE, April 1998.
- [TOT⁺99] Toshiyuki Takahashi, Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Hiroshi Harada Yutaka Ishikawa, and Pete H. Beckman. Implementation and Evaluation of MPI on an SMP Cluster. In *Parallel and Distributed Processing – IPPS/SPDP'99 Workshops*, volume 1586 of *Lecture Notes in Computer Science*, pages 1178–1192. Springer-Verlag, April 1999.

- [TSH⁺00] Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa. PM2: A High Performance Communication Middleware for Heterogeneous Network Environments. In *Supercomputing 2000, IEEE and ACM SIGARCH, November, 2000, (Published by CD-ROM)*., November 2000.
- [VIA] THE SPECIFICATION FOR THE VIRTUAL INTERFACE ARCHITECTURE:
<http://www.viarch.org/>.

List of Publications by the Author

Articles on which this dissertation is based

- 1 Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: GigaE PM: a High Performance Communication Facility using a Gigabit Ethernet, *New Generation Computing*, Springer-Verlag, Vol. 18, pages 177–186, January, 2000.
- 2 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: GigaE PM: The Design and Evaluation of High Performance Communication Facility Using a Gigabit Ethernet, *Transactions of Information Processing Society of Japan*, Information Processing Society of Japan, Vol. 41, Number 5, pages 1390–1399, May, 2000. (in Japanese)
- 3 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: High Performance Communication for Cluster Systems using an Existing Operating System Framework, *Transactions of Information Processing Society of Japan*, Information Processing Society of Japan, Vol. 41, Number 6, pages 1688–1696, June, 2000. (in Japanese)
- 4 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: The Design and Evaluation of High Performance Communication Facility: PM2, *Transactions of Information Processing Society of Japan*, Information Processing Society of Japan, Vol. 41, Number SIG 5(HPS 1), pages 80–90, August, 2000. (in Japanese)

- 5 Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: The Design and Evaluation of High Performance Communication using a Gigabit Ethernet, International Conference on Supercomputing '99(ICS'99), ACM SIGARCH, pages 243 – 250, June, 1999.
- 6 Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: High Performance Communication using a Commodity Network for Cluster Systems, in the Ninth International Symposium on High Performance Distributed Computing (HPDC-9), IEEE, pages 139–146, August, 2000.
- 7 Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa: PM2: A High Performance Communication Middleware for Heterogeneous Network Environments, Supercomputing 2000, IEEE and ACM SIGARCH, November, 2000, (Published by CD-ROM).
- 8 Yutaka Ishikawa, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, F. O'Carroll, and Hiroshi Harada: RWC PC Cluster II and SCORE Cluster System Software – High Performance Linux Cluster, In *Proceedings of the 5th Annual Linux Expo*, pages 55 – 62, 1999.
- 9 Toshiyuki Takahashi, Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Hiroshi Harada, Yutaka Ishikawa, and Pete H. Beckman: Implementation and Evaluation of MPI on an SMP Cluster, IPPS'99 2nd Workshop on Personal Computer Based Networks of Workstations, pages 1178 – 1192, 1999.

Other articles on which this dissertation is based

- 1 Hiroshi Harada, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, and Yutaka Ishikawa: SCASH:Software DSM using High performance network on commodity hardware and software, ACM Eighth Workshop on Scalable Shared-memory Multiprocessors, pages 26–27, 1999.
- 2 Yasushi Fujiimoto, Yuan Bin, Hisao Taoka, Hiroshi Tezuka, Shinji Sumimoto and Yutaka Ishikawa: Design and Implementation of a Real-Time Power System Simulator using a PC Cluster, 3rd Intl. conf. on Digital Power System Simulator, IEEE, May, 1999, (no page number).
- 3 Yasushi Fujiimoto, Yuan Bin, Hisao Taoka, Hiroshi Tezuka, Shinji Sumimoto and Yutaka Ishikawa: Real-time Power System Simulator on a PC Cluster, Intl. conf. on Power Systems Transients, pp 671–676, June, 1999.
- 4 Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Shinji Sumimoto, Toshiyuki Takahashi, and Hiroshi Harada: Parallel C++ Programming System on Cluster of Heterogeneous Computers, IPPS'99 Heterogeneous Computing Workshop '99,pages 73–82, April, 1999.
- 5 Hiroshi Harada, Yutaka Ishikawa, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, and Toshiyuki Takahashi: Dynamic Home Node Reallocation on Software Distributed Shared Memory, HPC Asia '2000, pages 158–163, May 2000.
- 6 Hiroshi Harada, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, and Yutaka Ishikawa: Comparison of Page Transfer Method on Software Distributed Shared Memory System, Transactions of Information Processing Society of Japan, Information Processing Society of Japan, Vol. 41, Number 5, pages 1410–1419, May, 2000. (in Japanese)

- 7 Shinji Sumimoto, Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, and Toshiyuki Takahashi: Design of High Performance Communication Library on Gigabit Ethernet, Special Interest Group Report of Information Processing Society of Japan 98-HPC-72 (SWoPP'98), Information Processing Society of Japan, pages 109–114, August, 1998, (in Japanese).
- 8 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: Performance Evaluation of Gigabit Ethernet NIC, Special Interest Group Report of Information Processing Society of Japan 99-HPC-75 (HOKKE'99), Information Processing Society of Japan, pages 49–54, March, 1999, (in Japanese).
- 9 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: The Design and Evaluation of High Performance Communication Library Using a Gigabit Ethernet, Proceedings of JSPP'99, Information Processing Society of Japan, pages 63 – 70, June, 1999, (in Japanese).
- 10 Shinji Sumimoto, Atsushi Hori, Hiroshi Tezuka, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa: GigaE PM II: Design of High Performance Communication Library using Gigabit Ethernet, Special Interest Group Report of Information Processing Society of Japan 99-ARC-134 (SWoPP'99), Information Processing Society of Japan, pages 61–66, August, 1999, (in Japanese).
- 11 Atsushi Hori, Hiroshi Tezuka, Toshiyuki Takahashi, Shinji Sumimoto, Noriyuki Soda, Hiroshi Harada, and Yutaka Ishikawa: Programming Environment for Clusters – SCORE Cluster System Software –, Special Interest Group Report of Information Processing Society of Japan 99-HPC-77 (SWoPP'99), Information Processing Society of Japan, pages 83–88, August, 1999, (in Japanese).

- 12 Noriyuki Soda, Hiroshi Tezuka, Shinji Sumimoto, Atsushi Hori, and Yutaka Ishikawa: Porting and Evaluation of PM communication library on UDP Special Interest Group Report of Information Processing Society of Japan 99-HPC-75 (HOKKE'99), Information Processing Society of Japan, pages 127–132, March, 1999, (in Japanese).
- 13 Hiroshi Harada, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, and Yutaka Ishikawa: Evaluation of Software Distributed Shared Memory System on Myrinet, Special Interest Group Report of Information Processing Society of Japan 98-HPC-73, Information Processing Society of Japan, pages 73–78, October, 1998, (in Japanese).
- 14 Hiroshi Harada, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, and Yutaka Ishikawa: Implementation and Evaluation of Memory Barrier on Software Distributed Shared Memory on Myrinet Proceedings of JSPP'99, Information Processing Society of Japan, pages 237 – 244, June, 1999, (in Japanese).
- 15 Hiroshi Harada, Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Shinji Sumimoto, and Toshiyuki Takahashi: Implementation and Evaluation of dynamic page manager node reallocation mechanism on SCASH Software Distributed Shared Memory Special Interest Group Report of Information Processing Society of Japan 99-HPC-77 (SWoPP'99), Information Processing Society of Japan, pages 89–94, August, 1999, (in Japanese).
- 16 Toshiyuki Takahashi, Francis O'Carroll, Atsushi Hori, Hiroshi Tezuka, Shinji Sumimoto, Hiroshi Harada, and Yutaka Ishikawa: Implementation and Evaluation of MPI on a SMP Cluster, Special Interest Group Report of Information Processing Society of Japan 98-HPC-72 (SWoPP'98), Information Processing Society of Japan, pages 115–120, August, 1998, (in Japanese).

Other articles which the author has written

- 1 Shinji Sumimoto: The Implementation and Evaluation of Shared Filesystem for Cluster Systems: Special Interest Group Report of Information Processing Society of Japan 95-HPC-70 (SWoPP'95), Information Processing Society of Japan, pages 9–16, August, 1995, (in Japanese).
- 2 Shinji Sumimoto: “Design and Evaluation of Fault-Tolerant Shared File System for Cluster Systems.” The Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing, IEEE, pages 74 – 83, June, 1996.

Appendix A

Performance and Costs of Commodity Hardware

A.1 CPU Performance and Memory Performance

Table A.1: SPEC95, System Call Cost and Memory Copy Performance

Year	Microprocessor		SPEC95		System	Memory
			int	fp	Call Cost	Copy
1995.	Pentium PRO	200 MHz	8.2	6.8	1.9 μs	52.3 MB/s
1997.	Pentium II	300 MHz	11.9	8.6	1.6 μs	100.0 MB/s
1998.	Pentium III	500 MHz	20.6	14.7	1.0 μs	140.0 MB/s
1999.	Pentium III	800 MHz	38.9	32.5	0.46 μs	170.0 MB/s

Table A.1 shows the SPEC95 benchmark results[SPE], system call cost, and memory copy performance on several Intel processor based systems. The SPEC95 benchmark is a benchmark suite designed to measure and compare computer performance across different hardware platforms developed by the Open Systems Group (OSG). OSG includes more than 30 computer vendors, systems integrators, publishers and consultants, and is part of the Standard Performance Evaluation Corp (SPEC). SPEC95 comprises two suites of benchmarks: SPECint95 for computation-intensive integer performance measurement and SPECfp95 for computation-intensive floating point performance measurement. The SPECint95 and SPECfp95 result numbers are based on the results achieved on a Sun S-PARCstation10/40 with 128 MB of memory and are both taken as "1." The system call cost results are measured in order to evaluate kernel trap overhead using a getpid() system

call which gets the process id. The memory copy performance was measured using a 10 MB memory copy to minimize CPU cache effect.

The SPEC95 and memory copy performance results in Table 2.1 show that CPU and memory performance improve as the CPU clock speed increases. Table 2.1 also shows that the system call cost decreases as the CPU clock speed increases.

A.2 Hardware Interrupt Costs

Table A.2: Hardware Interrupt Cost

Year	Microprocessor		Interrupt Cost
1996.	Pentium	150 MHz	5.6 μs
1997.	Pentium II	400 MHz	6.5 μs
1998.	Pentium III	500 MHz	5.9 μs

Table A.2 shows interrupt costs on several Intel processor-based systems. These costs are given by the time to reach the device driver routine after the NIC writes to the processor interrupt register on the Linux operating system. The results in Table 2.1 show that the interrupt cost does not decrease when the CPU clock speed increases.

A.3 I/O Bus Performance

Table A.3 shows the PCI DMA performance of a Myrinet NIC[MYR] on several Intel processor-based systems using Intel 440FX, 440BX, ServerSet III LE, and Intel i840 chipsets. These results were measured using the PCI DMA between host memory and Myrinet memory. The PCI DMA is triggered by the host CPU. Table A.3 shows that both the Intel 440FX and 440BX chipsets achieve almost maximum performance on a 32bit 33MHz PCI bus at a 64KB data transfer rate and that the ServerSet III LE achieves maximum performance on a 64bit 33MHz PCI bus. However, on a 64bit 66MHz PCI bus, PCI DMA performance depends on the chipset. The ServerSet III LE achieves over 500MB/s of bandwidth, however the Intel i840 chipset achieves 298MB/s of bandwidth at a 64KB data transfer rate from NIC to host memory, and only 159MB/s of bandwidth at a 64KB

data transfer rate from host to NIC memory. Table A.3 also shows that the PCI DMA performances achieved at a 1KB data transfer rate are around 80-90 MB/s, even on a 64bit 66MHz PCI bus is used.

Table A.3: PCI DMA Performance

Chipset	PCI	Directions	PCI DMA Message Size (MB/s)				
			1KB	2KB	4KB	8KB	64KB
Intel 440FX	32bit, 33MHz	Host → NIC	82.7	115.9	125.2	128.0	130.7
		NIC → Host	82.5	119.7	127.5	130.5	133.4
Intel 440BX	32bit, 33MHz	Host → NIC	84.8	87.2	115.8	117.5	120.7
		NIC → Host	84.7	118.7	125.6	129.5	132.2
ServerSet III LE	64bit, 33MHz	Host → NIC	70.2	140.3	134.8	179.0	260.2
		NIC → Host	66.1	131.2	197.5	226.8	259.7
ServerSet III LE	64bit, 66MHz	Host → NIC	88.0	175.9	352.1	437.4	505.4
		NIC → Host	84.7	169.5	338.8	475.0	522.6
Intel i840	64bit, 66MHz	Host → NIC	81.3	86.6	117.2	143.4	158.4
		NIC → Host	79.1	157.2	259.1	279.1	298.2

Evaluation Environments:

Intel 440FX: Pentium PRO 200MHz, 32bit Myrinet NIC

Intel 440BX: Pentium III 500MHz, 32bit Myrinet NIC

ServerSet III LE: Pentium III 800MHz, 64bit Myrinet NIC

Intel i840: Pentium III Xeon 733MHz, 64bit Myrinet NIC

Appendix B

GigaE PM Protocol

In this appendix, the GigaE PM network protocol is described in detail. There are sending and receiving buffers for each channel. All out-going messages are stored in the sending buffer and all in-coming messages are stored in the receiving buffer. Unlike TCP/IP, buffers are not allocated with respect to peer to peer communication, but are allocated with respect to a channel.

A message is represented by $Msg(SenderID, ReceiverID, DataMessageSequenceNumber)$. Let N represent the number of messages that the sender may send asynchronously without waiting for an ACK. Timeout is referred to as T . Let $SBuf(r, i)$ represent the message buffer where the i th message is sent to the receiver r . Let $STime(r, i)$ represent the time when the i th message is sent to the receiver r . $MsgIdSent(r)$ keeps the largest sequence number of a message which has been sent to the receiver r . $MsgIdRecv(s)$ keeps the largest sequence number of a message sent by the sender s and received by the receiver. $MsgIdAcked(r)$ keeps the largest sequence number of an ACK message received from the receiver r . At initialization, $MsgIdSent(r)$, $MsgIdRecv(s)$ and $MsgIdAcked(r)$ are 0.

The GigaE PM protocol on sender and receiver nodes is described as follows:

On a Sender Node:

S1 Sender s may send receiver r messages $\forall Msg(s, r, i)$ where

$$MsgIdSent(r) < i < MsgIdAcked(r) + N$$

For each message $Msg(s, r, i)$, the following procedure is performed:

- 1 The $Msg(s, r, i)$ is sent to the receiver.
- 2 The sender message buffer $SBuf(r, i)$ of $Msg(s, r, i)$ is created and kept.
- 3 The current time is kept in $STime(r, i)$.
- 4 $MsgIdSent(r) \leftarrow MsgIdSent(r) + 1$.

S2 When the difference between the current time and $STime(s, r, i)$ is larger than T , the following procedure is performed:

- 1 $MsgIdSent(r) \leftarrow i$.
- 2 Perform S1.

- If a LOSE message $LOSE(r, k)$ where

$$MsgIdAcked(r) < k < MsgIdAcked(r) + N$$

is received,

release the sender message buffer, $\forall SBuf(r, i)$ where $MsgIdAcked(r) < k$.

- 1 $MsgIdSent(r) \leftarrow k$.
- 2 $MsgIdAcked(r) \leftarrow k$.
- 3 Perform S1.

- If a STOP message $STOP(r, k)$ where

$$MsgIdAcked(r) < k < MsgIdAcked(r) + N$$

is received,

- 1 release the sender message buffer, $\forall SBuf(r, i)$ where $MsgIdAcked(r) < k$.
- 2 $MsgIdAcked(r) \leftarrow k$.
- 3 Stop sending.

- If a GO message $GO(r, k)$ where

$$MsgIdAcked(r) \leq k < MsgIdAcked(r) + N$$

is received, perform S1.

- If an ACK message $ACK(r, k)$ where
 $MsgIdAcked(r) \leq k < MsgIdAcked(r) + N$
is received,

- 1 $MsgIdAcked(r) \leftarrow k$.

- 2 Perform S1.

On a Receiver Node:

- When receiver r receives message $Msg(s, r, i)$,
 - In the case where $MsgIdRecv(s) + 1 = i$ and the receiver buffer on the host is not full,
 - An ACK message $ACK(r, i)$ is sent back to sender s .
 - The message is transferred to the host.
 - $MsgIdRecv(s) \leftarrow MsgIdRecv(s) + 1$.
 - In the case where $MsgIdRecv(s) + 1 = i$ and the receiver buffer on the host is full,
 - A STOP message $STOP(r, i)$ is sent back to sender s .
 - All subsequent received messages are discarded.
 - In the case where $MsgIdRecv(s) + 1 < i$ which means that a message has been lost, a LOSE message $LOSE(r, MsgIdRecv(s))$ is sent back to the sender s .
- When the receive buffer on the host again has room and a STOP message has been sent, A GO message $GO(r, MsgIdRecv(s) + 1)$ is sent back to sender s . Then receiver r can again receive messages.

Appendix C

NAS Parallel Benchmarks

The Numerical Aerospace Simulation (NAS) Parallel Benchmarks[NPB] are a set of 8 programs designed to evaluate the performance of parallel supercomputers. The NPB was developed by the Numerical Aerospace Simulation Systems Division of the NASA Ames Research Center in order to provide the Nation's aerospace research and development community a high-performance, operational computing system capable of simulating an entire aerospace vehicle system within a computing time of one to several hours. The programs of the NPB suite are based on computational fluid dynamics (CFD) applications. The NPB programs are written using MPI, so, the NPB programs can be executed on any system which supports MPI. The NPB programs have several classes which depend on data size, called class S, W, A, B, C and D. Class S uses the smallest data size and class D the largest. All of the NPB programs, except IS, execute floating point calculations, and the communication patterns of each benchmark program are different from the others.

Table C.1 shows the NPB programs and their dominant message characteristics. Class A 16 node programs are used to show examples of message communication patterns: approximate message sizes, number of messages and execution time on a 16 node cluster with a Pentium III 500MHz processor and Myrinet. These results are based on the same histogram data on which the [TSH⁺00] paper was based. The message sizes and number of messages are different based on the number of nodes and the class.

Table C.1 shows that the IS, CG and FT programs are sensitive to communication bandwidth and latency. The EP, LU, MG, BT and SP programs are not sensitive to communication performance when compared with IS, CG and FT [KISB99, SHT⁺00b, TSH⁺00].

Table C.1: NAS Parallel Benchmarks and Dominant Message Characteristics on Class A, 16 Node programs

Name	Note	Message Characteristics on Class A, 16 node		
		Dominant Sizes of Messages	Number of Messages	Execution Time
CG	Conjugate Gradient	16 - 32 KBytes	1248 / PE	3.92 sec.
EP	Embarrassingly Parallel	8 - 16 Bytes	12 / PE	24.18 sec.
FT	Fourier Transform	512 - 1024 KBytes	128 / PE	14.11 sec.
IS	Integer Sort	64 - 256 KBytes	176 / PE	1.68 sec.
LU	LU Decomposition	512 - 1024 Bytes	46000 / PE	77.97 sec.
MG	Multi-Grid	8 Bytes - 256 KBytes	832 / PE	5.17 sec.
BT	Block Tridiagonal Solver	16 - 128 KBytes	4800 / PE	161.89 sec.
SP	Pentadiagonal Solver	16 - 64 KBytes	9600 / PE	115.01 sec.

The IS and FT programs use All-to-All collective communication. The dominant message size of the FT Class A program is in the 512–1024 KBytes range on a 16 node cluster, that of IS is in the 64–256 KBytes range. The CG program uses neighbor to neighbor communication and sends a number of messages continuously. The dominant message size of CG Class A is in the 16–32 KBytes range. The LU program also uses neighbor to neighbor communication and sends messages periodically. The dominant message size of LU Class A is in the 512–1024 Bytes range.

The NPB results on several commercial parallel computers are published on the NPB web page (<http://www.nas.nasa.gov/Software/NPB/>). The NPB results have been widely measured on a number of parallel machines, and published in various papers. So, researchers can compare their performance with that of others.

Appendix D

NPB Results on RWC SCore Cluster I

Figures D.1, D.2, D.3, D.4, D.5, D.6, D.7 and D.8 show the results of tests using the NAS parallel benchmark on RWC SCore Cluster I.

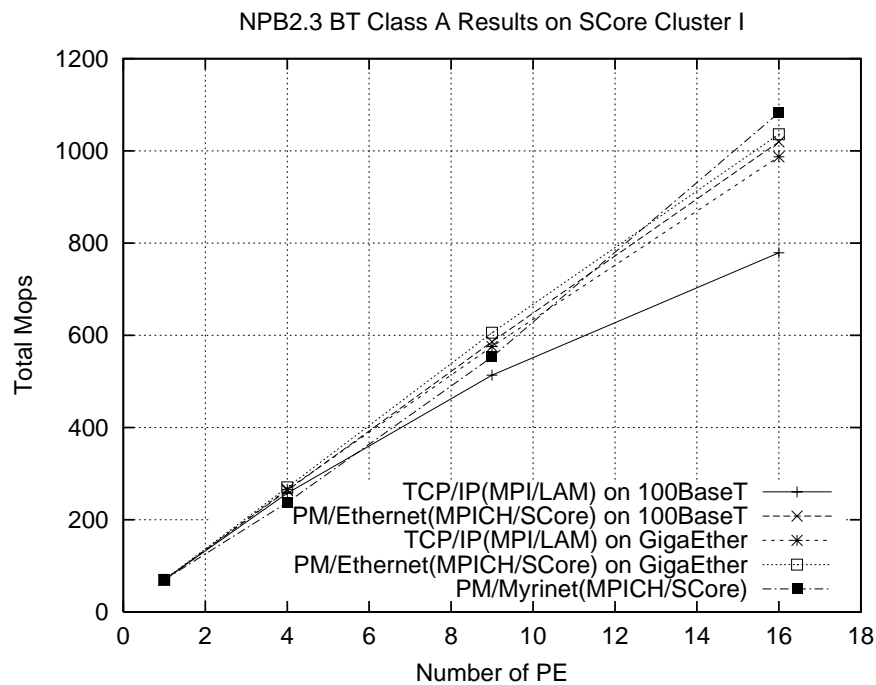


Figure D.1: NPB BT CLASS A on RWC SCore Cluster I

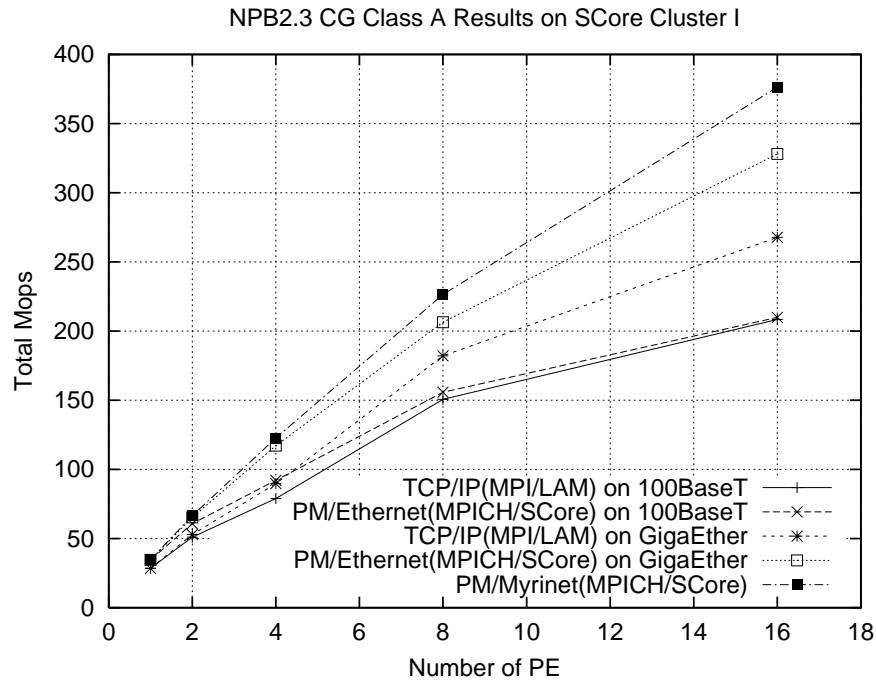


Figure D.2: NPB CG CLASS A on RWC SCore Cluster I

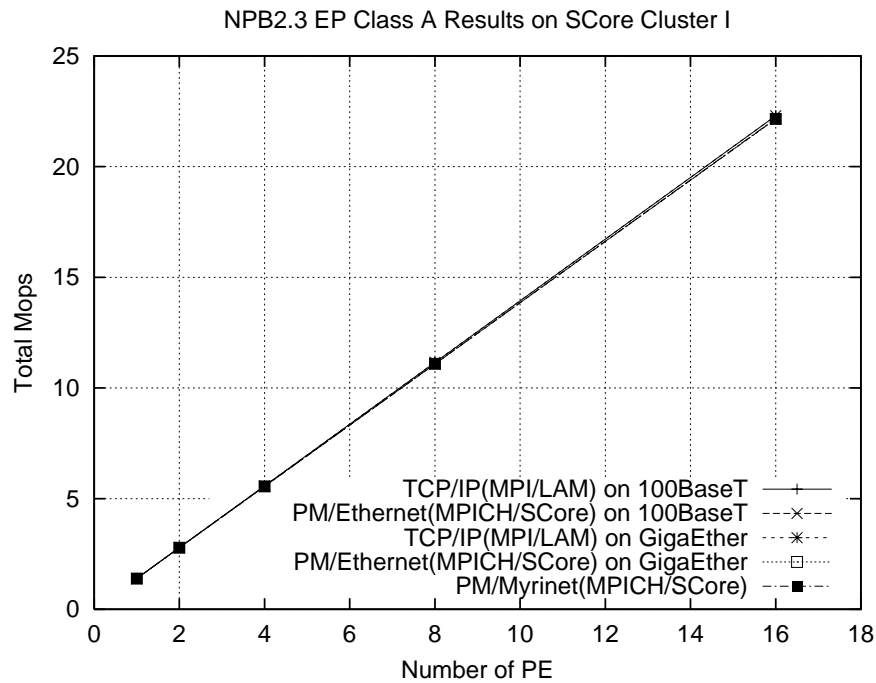


Figure D.3: NPB EP CLASS A on RWC SCore Cluster I

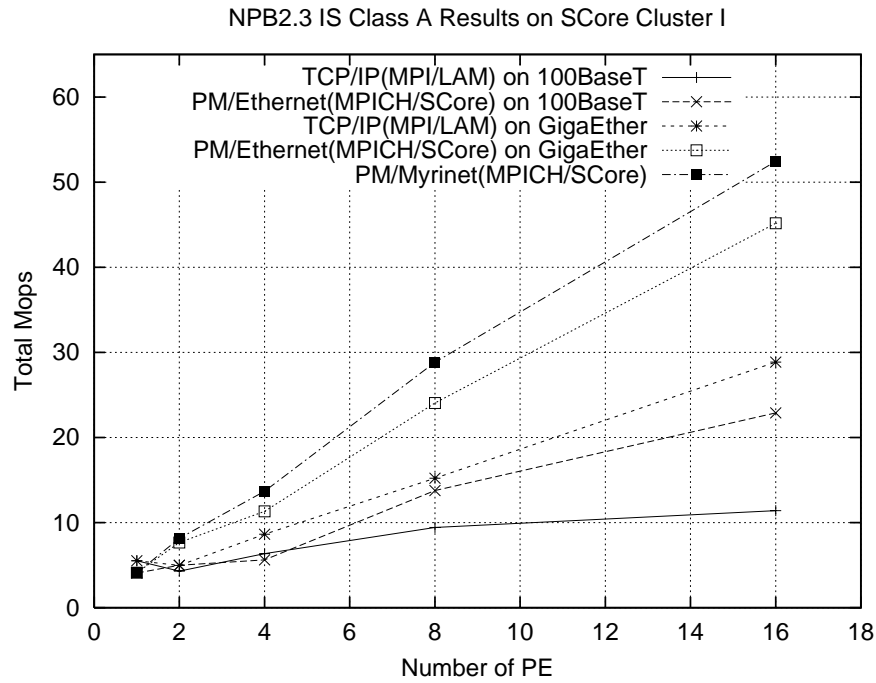


Figure D.4: NPB IS CLASS A on RWC SCore Cluster I

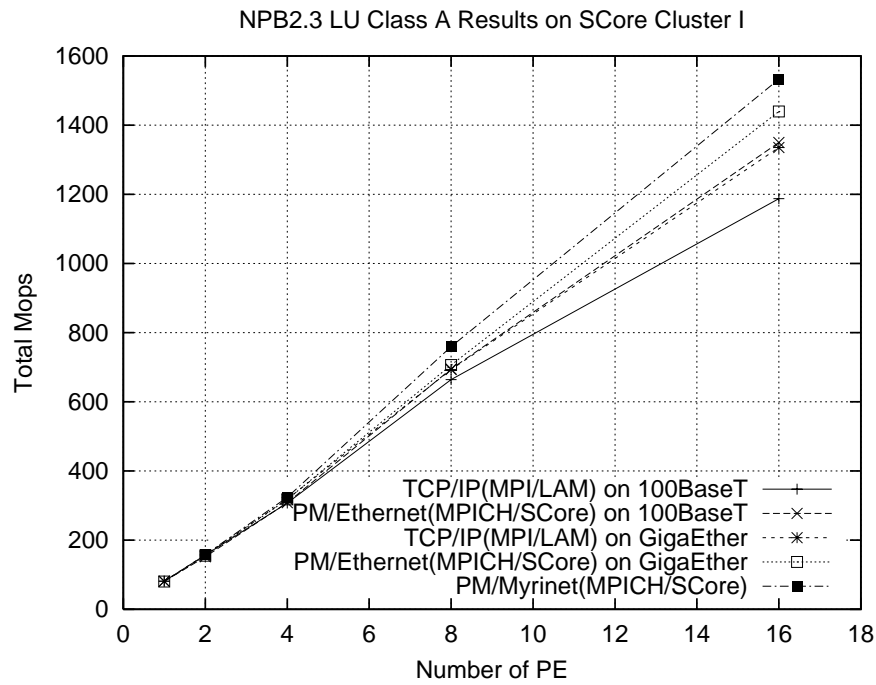


Figure D.5: NPB LU CLASS A on RWC SCore Cluster I

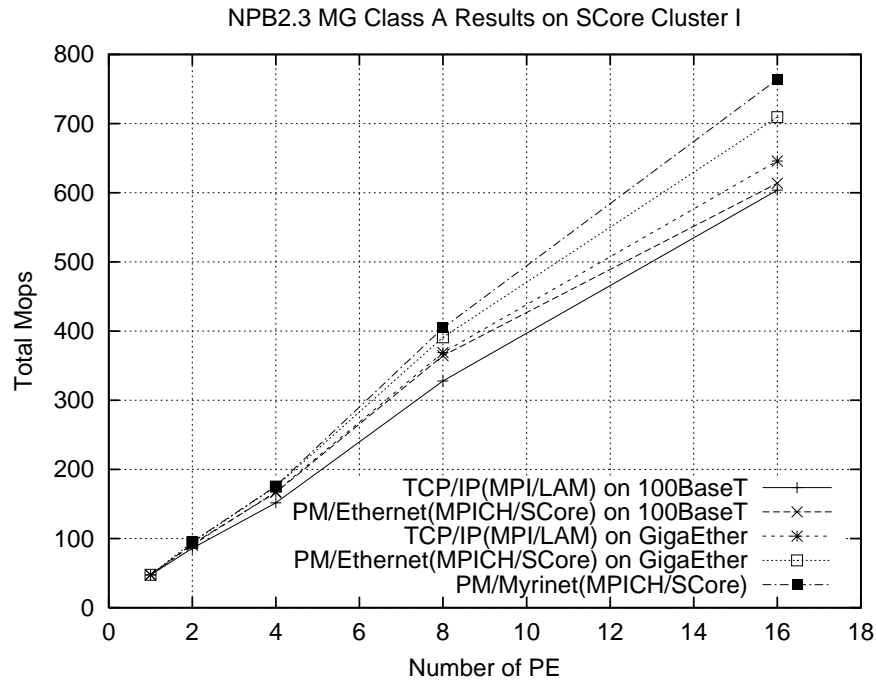


Figure D.6: NPB MG CLASS A on RWC SCore Cluster I

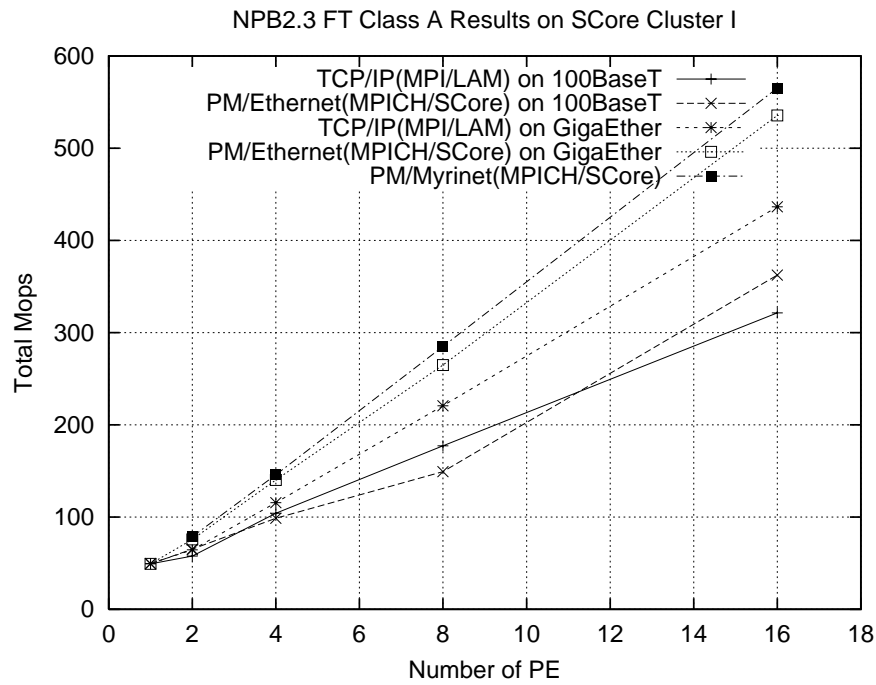


Figure D.7: NPB FT CLASS A on RWC SCore Cluster I

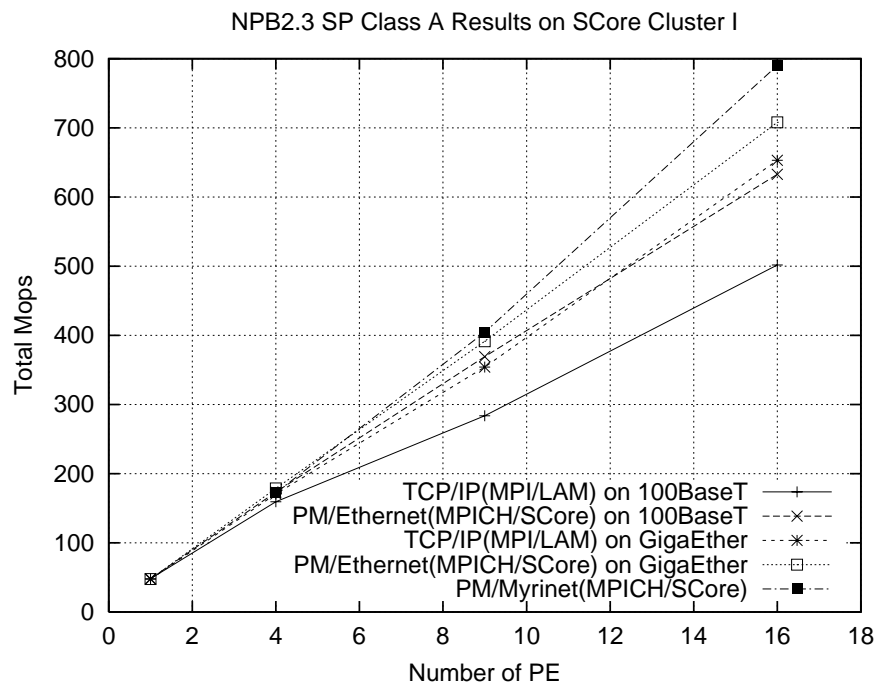


Figure D.8: NPB SP CLASS A on RWC SCore Cluster I

Appendix E

NPB Results on RWC PC Cluster II

Figures E.1, E.2, E.3, E.4, E.5, E.6, E.7 and E.8 show the results of tests using the NAS parallel benchmark on RWC PC Cluster II.

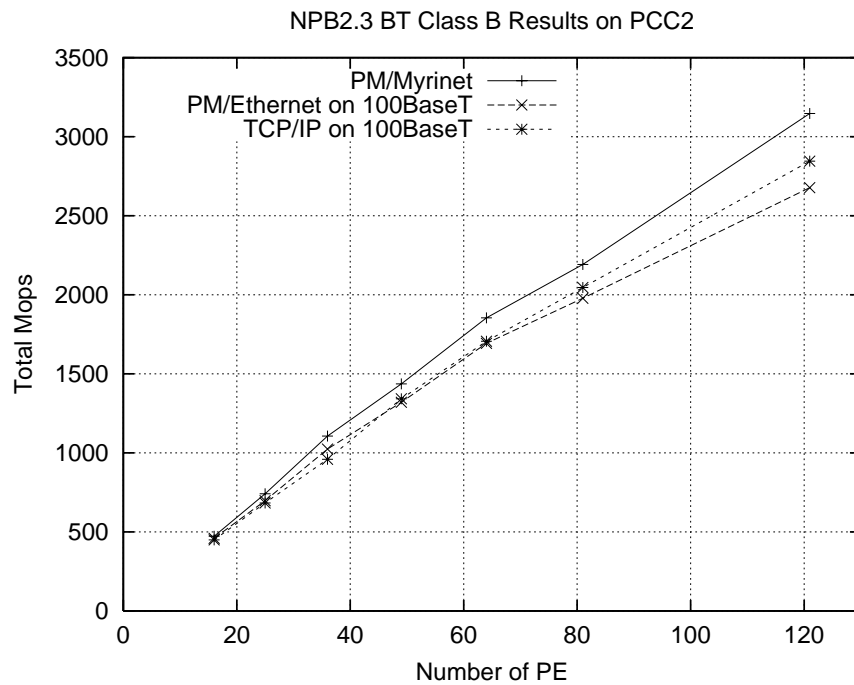


Figure E.1: NPB BT CLASS B on RWC PC Cluster II

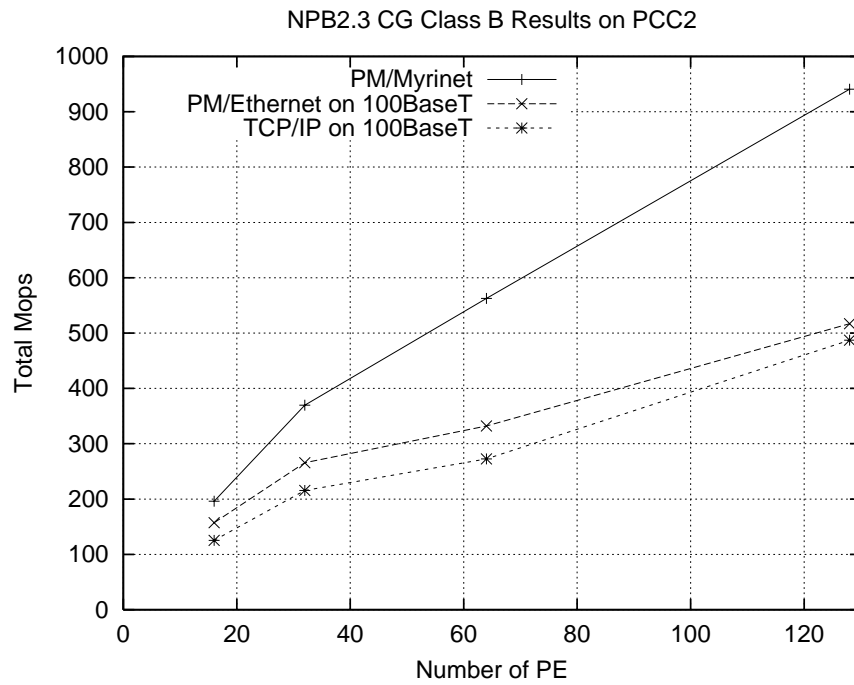


Figure E.2: NPB CG CLASS B on RWC PC Cluster II

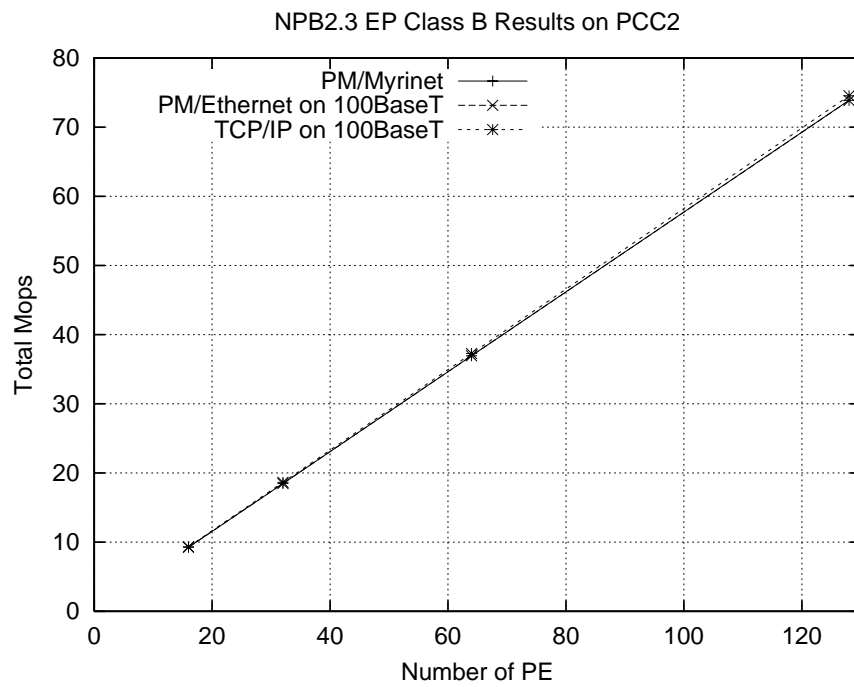


Figure E.3: NPB EP CLASS B on RWC PC Cluster II

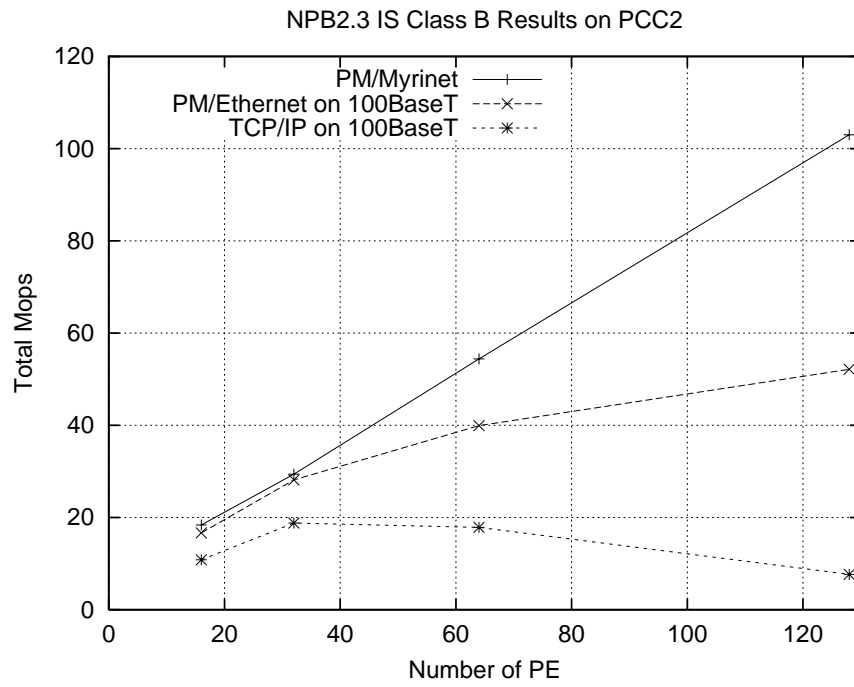


Figure E.4: NPB IS CLASS B on RWC PC Cluster II

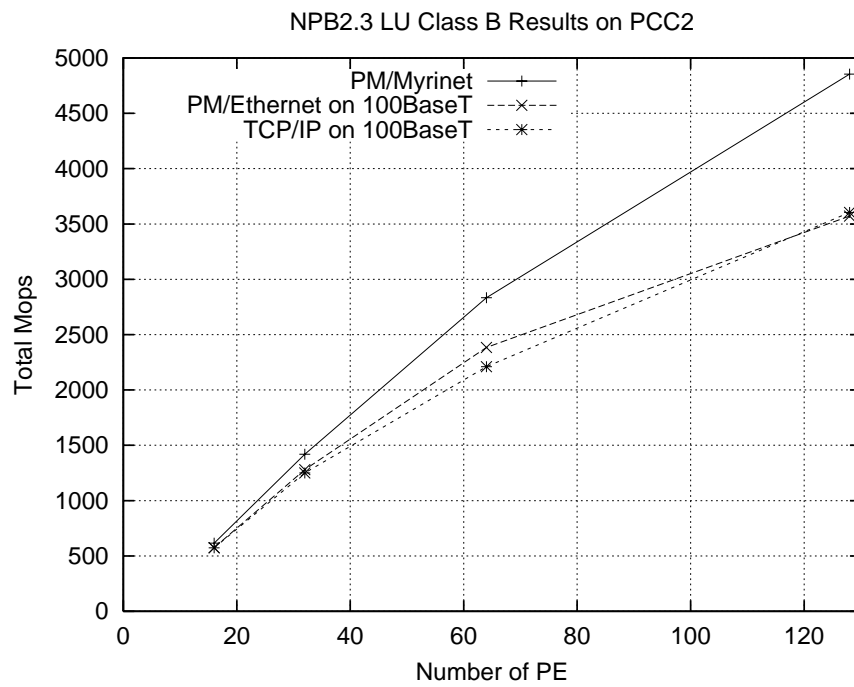


Figure E.5: NPB LU CLASS B on RWC PC Cluster II

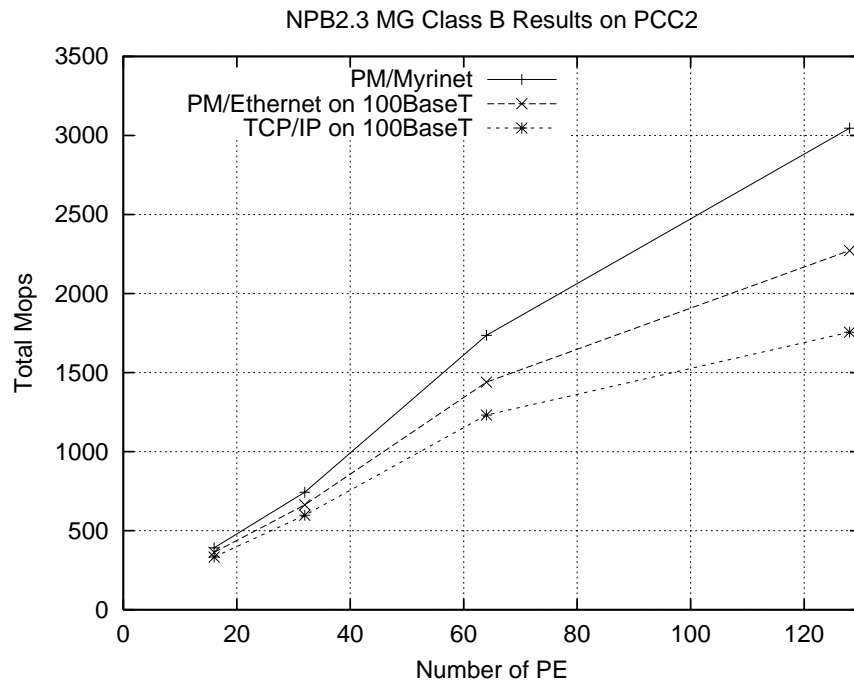


Figure E.6: NPB MG CLASS B on RWC PC Cluster II

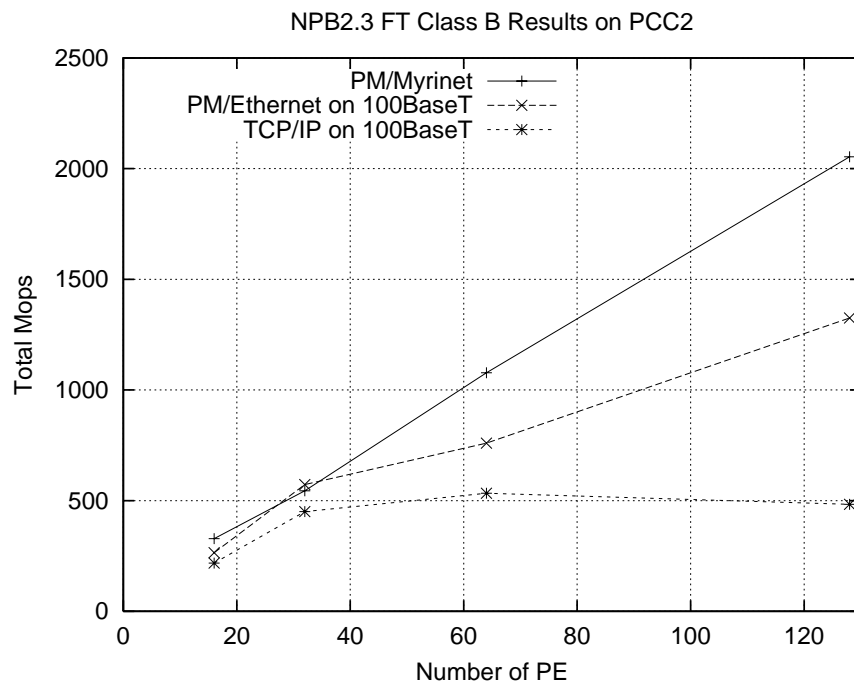


Figure E.7: NPB FT CLASS B on RWC PC Cluster II

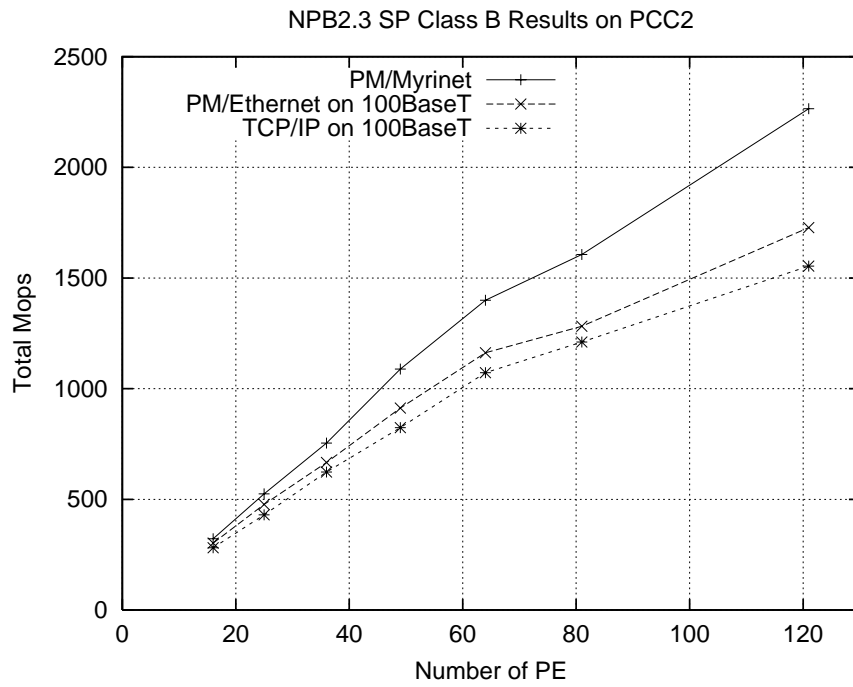


Figure E.8: NPB SP CLASS B on RWC PC Cluster II