Performance Evaluation and Analysis of User Level Networking Protocols in Clusters

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

N. Sencer Kutluğ, B.S.

* * * * *

The Ohio State University

2000

Master's Examination Committee:

Prof. Dhabaleswar K. Panda, Adviser

Prof. Ponnuswamy Sadayappan

Approved by

Adviser

Department of Computer
and Information Science

Performance Evaluation and Analysis of User Level Networking Protocols in Clusters

By

N. Sencer Kutluğ, M.S.

The Ohio State University, 2000

Prof. Dhabaleswar K. Panda, Adviser

In recent years, clusters of workstations have become a viable alternative to supercomputers for high performance computing. With the introduction of high speed networking hardware in *Local Area Networks*, the performance bottleneck in clusters of workstations has shifted from networking hardware to communications protocols that provide an abstract interface to the hardware. Recently, a set of communication protocols such as AM, VMMC, FM, U-Net, BIP, and more recently VIA have been proposed to alleviate this bottleneck. These protocols bypass the operating system and avoid extra copies of data for better performance. Because of minimal kernel involvement, these protocols are usually referred as *User Level Networking Protocols* (ULNP). ULNPs are not suitable for writing parallel programs, and another level of abstraction is used between the user level networking protocols and the application software. This layer is typically called a programming environment. One of the

most widely used parallel programming environment layers is the *Message Passing Interface* (MPI).

As new ULNPs are developed and better implementations of MPI on top of ULNPs become available , there is a need for a comprehensive benchmark suite which can provide performance evaluation and insights into these implementations. In this thesis, we develop a set of micro-benchmarks to evaluate the performance of ULNPs and MPI implementations on top of ULNPs. The first set of micro-benchmarks are provided at the ULNP level. These benchmarks provide comprehensive performance evaluation of ULNPs. A second set of micro-benchmarks are provided at the MPI level, and is an extension of the first set of micro-benchmarks to the higher layer. By using these micro-benchmarks, we are able to analyze how the performance issues at the lower layer can affect the performance at the higher layers. These two sets of micro-benchmarks can also be used to determine whether the source of communication performance bottleneck lies at the ULNP layer or at the MPI layer. Using our micro-benchmarks we evaluate the performance of some implementations of ULNPs, and the corresponding MPI implementations. We also evaluate the performance of the NAS benchmarks and attempt to establish correlation between the performance obtained at the ULNPs, at the MPI layer, and at the application layer. The results indicate a strong correlation between these performance results.

ACKNOWLEDGMENTS

I would like to acknowledge the guidance and support of my advisor Professor D. Panda without whom this project would have never been possible. I would like to acknowledge for the financial support from the National Science Foundation and the Ohio Board of Regents awarded to Prof. D. Panda provided to me as a Research Assistant for this thesis work. I would also like to acknowledge Prof. P. Sadayappan for his involvement in this work and for his invaluable suggestions.

The fellow research students, Mohammad Banikazemi and Darius Buntinas have also contributed in-terms of insightful discussions.

I would like to thank my family for all the support they have given me during my studies.

Last, but not least, I would like to acknowledge my friends for being there whenever I needed them.

VITA

February 6, 1975 ..... Born - Ankara, Türkiye

1998 ..... B.S., Computer Engineering,
Boğaziçi University, Istanbul, Türkiye.

Fall 1999 - Spring 2000 ..... Graduate Research Associate,
The Ohio State University.

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in:

Computer Architecture	Prof. Dhabaleswar K. Panda
Operating Systems	Prof. Mukesh Singhal
Programming Languages	Prof. Neelam Soundarajan
Analysis of Algorithms	Prof. Steve Lai
Theory of Computation	Prof. Ken Supowit

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, the computer industry has witnessed a dramatic improvement in the performance of PCs/workstations. Microprocessor performance has been improving at a rate of 50% per year and even an ordinary PC sitting on a desk in our home is capable of delivering better performance than supercomputers of a decade ago. Also, with the transformation of PCs/workstations into a commodity component, such as TV, the prices have come down significantly. Low prices of these commodity components accompanied by the vast improvement in performance have made clusters of workstations an attractive alternative to large scale *Massively-Parallel Processors* (MPP) for high performance computing [9].

The signature of distributed high performance applications is that they require a low latency, high bandwidth communication facility for exchanging data and synchronization operations. On the hardware side, raw bandwidth of networks has increased significantly in the past few years. Networking hardware supporting bandwidths in the order of gigabits per second have become widely available, such as Myrinet [14], ATM [10], Fibre-Channel [28], and FDDI [8]. However, with the introduction of high speed networking hardware in *Local Area Networks* (LAN), the performance bottleneck has shifted from networking hardware to communication protocols. These

protocols provide an abstract interfaces to the high speed networks. Because of their wide availability, TCP/IP or UDP/IP has been traditionally used for communication in cluster environments. However, high overheads of these protocols have been leading to poor performance in clusters. One way message latency for TCP/IP is two orders of magnitude higher than the latencies obtained by lightweight protocols used in MPPs. Clearly, this is not acceptable for developing and deploying distributed high performance applications on clusters.

1.1 User Level Networking Protocols

Poor performance of the legacy protocols is a consequence of several sub-steps involved in protocol processing. These include protocol overhead, buffer management, link management, operating system overhead, making multiple copies of data, and kernel intervention in the data communication critical path. In recent years, the research community has proposed a group of new communication systems/protocols [19] such as AM [32], VMMC [13], FM [26], U-Net [31, 33], and BIP [27] to address these issues. All of these communication systems use much simpler communication protocols in comparison with the legacy protocols such as TCP/IP. The role of the operating system has been much reduced in these systems and in most cases, user applications are given direct access to the network interface to send or receive messages. Because of minimal kernel involvement and the flexibility of transferring data from user address space, these protocols are typically referred as *User Level Networking Protocols* (ULNP). Recently, in order to provide a standard for user level network interfaces, some industry leaders have proposed the *Virtual Interface Architecture* (VIA) [7]. The VIA standard specifies only the *Application Programmers Interface*

(API) for user level networking protocols, however, the vendor is free to make any implementation decisions. The VIA standard has been influenced mostly by U-Net and VMMC.

1.2 Motivation and Approach

Since communication performance is very crucial to the success of cluster of workstations as solutions for high performance computing, it is important that ULNPs get implemented in the best possible manner. However, the performance delivered by the ULNP is not the performance seen by the application. Communication in cluster of workstations goes through several layers of abstractions. The performance delivered to the application is affected by each of these layers. As shown in Fig. 1.1, at the bottom we have the physical communication medium. On top of the networking hardware, ULNPs provide an abstraction to the hardware. However, for programming applications, ULNP API is very rudimentary. Thus, a programming environment layer is needed for easing the task of writing programs. In this hierarchical organization, each layer adds an overhead, and degrades the performance delivered by the layer below it.

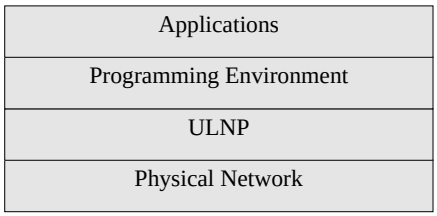

Figure 1.1: Layers of abstraction from Network to Application

Currently there is no comprehensive framework to evaluate such performance degradation. Such a framework, if developed, can be used to evaluate the performance of ULNPs, extend it to the programming environment layer, analyze the performance at the programming environment layer, and see how the performance is all carried up to the application layer. We take on such a challenge in this thesis. We develop a framework consisting of two sets of micro-benchmarks.

The first set of micro-benchmarks we propose for evaluating the ULNP are at the low level and are implemented on top of the ULNP layer. ULNPs have some characteristics that are different from that of local area networking protocols. Their implementation is more hardware and operating system dependent, and some design decisions in their implementations might affect the performance significantly. One should realize that in the case of these ULNPs, even a couple of microseconds is considered significant since they might constitute a large portion of the latency. For example, in Giganet CLAN VIA [1], one way latency is only 9μseconds. Since kernel involvement is minimized in ULNPs, some of the functions provided by the kernel in the legacy protocols should be implemented within the ULNPs. Depending on the implementation, the performance delivered might be very different under different environments. For example, in most of the ULNPs messages are delivered directly from the the user address space to the *Network Interface* (NI) buffers. Since the user uses virtual addresses and network interface uses physical addresses, there is a need for an address translation service. This service is provided by the kernel in the legacy protocols, but this is not the case with the ULNPs. Furthermore, since these protocols are developed to be used in *System Area Networks* (SAN) for high performance applications, they provide some services which are not provided by legacy

protocols, such as support for sending from or receiving to non-contiguous buffers. Our proposed micro-benchmarks evaluate the performance of these services as well as the effect of the design decisions on the performance. From the point of view of the protocol implementors, the proposed benchmarks are useful in finding the bottlenecks, and identifying potential problems with the implementations otherwise not visible by the traditional performance evaluation approach such as using latency, bandwidth, or the LogP parameters [17]. For the users of these implementations, the benchmarks might help in determining which services of the protocol to use and which services to implement at the higher layers. This is done by showing the performances of these services.

Besides the low level communication abstraction, as mentioned earlier, a portable parallel programming environment [19] is the key to the success of high performance computing systems. Over the last few years, researchers have developed standard interfaces such as PVM [30, 34] and *Message Passing Interface* (MPI [15, 24, 25]) to provide portability. These interfaces and standards do not force an application developer to understand the intricate details of the hardware, software, and network characteristics. However, the performance of applications depends heavily on the latency and bandwidth required for interprocessor communication and synchronization across the nodes. Therefore, it is crucial to provide an efficient implementation of message passing interfaces such as MPI on top of the ULNPs. The second part of this thesis proposes a set of benchmarks parallel to those proposed earlier to measure the performance effects of the ULNP on the MPI implementation. This second set of benchmarks is implemented on top of MPI. Along with the first set of benchmarks they can be used to determine what percentage of the performance of the lower layer

is delivered to the user at the MPI level. These benchmarks might be used to assess the quality of a given MPI implementation and to determine whether the bottleneck is at the ULNP implementation or the MPI implementation.

Finally we use a commonly used set of benchmarks, namely NAS parallel benchmarks [4] from NASA Ames Research Center, to derive correlation between the performance of applications and the performance results obtained at the ULNP and MPI layers by using our benchmarking suite. Such an analysis clearly demonstrate performance degradation introduced by the ULNP and MPI layers.

1.3 Related Work

Traditionally, latency and bandwidth have been used for measuring the network performance. More recently, LogP [17] parameters have been proposed to gain insights about different components of a communication step. However, because of the distinct characteristics of the user level protocols, as discussed in the previous section, these performance evaluation methods do not provide a comprehensive picture of what performance an application might see while being executed on top of a given ULNP and MPI implementation.

Latency test is a ping-pong test where the sender sends a message and the receiver upon receiving it, immediately replies back with a same size message. By repeating this test sufficient number of times, and taking the average, the effect of transient conditions of the network are eliminated from the result. Latency test gives us the time to transmit messages between two communicating nodes.

The bandwidth test also consists of a sender and a receiver. In this test, the sender constantly pumps data into the network without waiting for an acknowledgement.

The receiver, upon receiving all the messages, sends back an acknowledgement. The sender concludes the test upon receiving the acknowledgement. The bandwidth test essentially measures the capacity of the network.

However, in the following chapters we argue that for getting a comprehensive picture, latency and bandwidth metrics are not sufficient because of the distinct features of user level networking protocols. Although latency and bandwidth give some idea about how the communication subsystem performs, it does not provide any information about how the performance might be affected under different conditions.

The LogP model [17] was proposed as a new model for parallel computation and parallel algorithm development. Besides computation cost, it also considers the communication cost, and the cost of integrating communication into computation, which have been ignored in the previous performance models. LogP parameters consist of four metrics that capture a parallel system. These four metrics are the number of processors(P), a communication bandwidth(g), the communication delay(L), and the communication overhead(o). There has been further refinements to these parameters by dividing the communication overhead into receive and send overheads [18]. The LogP parameters divide the communication path into several components, as shown in Fig. 1.2. O_s is the time the sender processor spends for sending a message. Similarly, O_r is the time the receiver processor spends for receiving a message. The parameter g is the time between sending/receiving of consecutive messages, and L is the time spent in the network.

The LogP model has been used for evaluating the performance of several user level networking protocols [18, 23]. Although it gives a better picture than the classical latency and bandwidth figures, the manner in which it has been used does not account

for a number of significant factors that affect the performance of ULNPs. For example, the effect of multiple communicating entities, transmission of successive messages from different buffers, impact of physically non-contiguous send buffers, etc. have not been modeled.

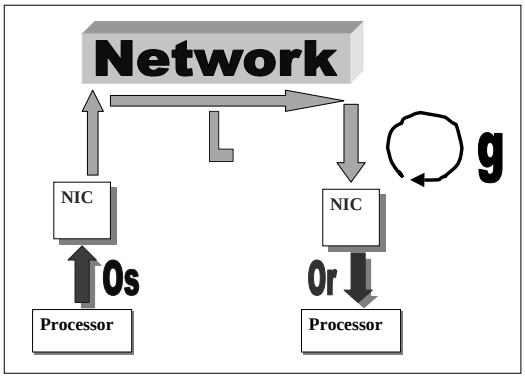

Figure 1.2: Breakdown of LogP parameters

1.4 Thesis Outline

In this thesis, a bottom up approach for evaluating user level networking protocols is proposed. In chapter 2, design issues in ULNPs which affected our choice of micro-benchmarks are discussed. Also, a survey of major user level networking protocols is provided in chronological order. The protocols that are thought to be relevant in the context of this work are included. Chapter 3 talks about proposed low level micro-benchmarks for evaluating the performance of ULNPs. By using these benchmarks, three VIA implementations and one GM implementation are evaluated and analysis of the results are given. Chapter 4 discusses another set of benchmarks similar to

those mentioned in Chapter 3, but implemented on top of MPI. By using these benchmarks, we evaluate three MPI implementations over different ULNPs. An analysis and comparison of the results are given in context of previous results. In Chapter 5, we go up even a layer higher, and compare the results obtained from our benchmark suite to the NAS parallel benchmark results, and show the correlation between the performance at lower layers, and the performance seen by an application. Finally, we conclude and discuss future directions in Chapter 6.

CHAPTER 2

USER LEVEL NETWORKING PROTOCOLS

In this chapter we give a brief overview of user level networking protocols as well as the design issues involved in implementing them. The ULNPs move some of the services normally provided by kernel into the user level. Thus, the designers of these protocols face many challenges and appropriate design decisions need to be taken. The decisions taken by the designers for the implementations directly affect the performance of the system. Thus, any performance evaluation framework needs to be aware of these design issues. In this chapter, we examine some user level networking protocols and see how they handle some of these design issues. This will provide us insight to the choice of micro-benchmarks.

2.1 Design Issues for ULNPs

2.1.1 Zero-Copy vs. True Zero-Copy

One of the main motivations behind ULNPs is to prevent unnecessary copying of data, thus improving message latency. In the legacy communication protocols, the communication goes through the kernel and messages have to be copied to the kernel buffers for communication to take place. Since user level networking protocols bypass the kernel in the critical path, user can manage its own buffers for communication

and going through the kernel buffers can be avoided. However, there are many issues that need to be resolved to provide zero-copying.

In the literature, there are two interpretations of the term "zero-copy". Both versions of zero-copy are better than going through the kernel, however one is better than the other in most of the cases.

In one implementation of a zero-copy protocol, user has to register some memory region before starting communication and all the messages should go through this registered memory region. There is an upper bound in the size of the registered region. These registered buffers are mapped to NIC's DMA address space so that the NIC can directly access these buffers for send and receive operations. This region is also a part of user application's memory and the application can form the messages in this region. However, in most of the cases the application needs to copy messages from somewhere else in its address space into the registered region, or for receiving messages it needs to copy the incoming messages to somewhere else in order to make room for more incoming messages. Going through kernel buffers is still avoided, but extra copies might still be required. Thus, this implementation uses a relaxed interpretation of "zero-copy".

On the other hand, true zero-copy lets user use send/receive messages to/from anywhere in its address space. However, providing this is quite difficult. First of all the user provides virtual addresses. But, for DMA operations NIC uses physical addresses. Thus, there needs to be some mechanism on the NIC for doing the translation from virtual to physical addresses. Also, since user can send from anywhere in its address space, NIC should be able to access all of the physical memory. Another complication arises from the requirement that for DMA operations the memory regions should

be pinned in the physical memory, since asynchronous page swapping operations of kernel may corrupt the data while it is being DMA'ed. Thus, dynamic pinning and unpinning of memory regions are necessary.

In the previous zero-copy we talked about, registered memory region is already pinned down at the registration time. Thus, dynamic pinning or handling page faults is not necessary. Also, for specifying a buffer for transfer the user specifies the offset within the registered memory region. Thus, there is no need for address translation. Such kind of zero-copying has been implemented in [26, 31, 32], and true zero-copying has been implemented in [33].

2.1.2 Division of Labor Between Host and Network Coprocessor

ULNPs have emerged as a consequence of smart NICs with memory and CPUs. In order to provide better performance, the ULNPs bypass the operating system, thus avoiding any costs associated with switching to the privileged mode. In the same time, there is a performance gain in the overall system performance because the host processor can be doing computation, while the communication is taken care of by the NIC. In ULNPs, the host and NIC share the load in transferring a message. However, in many real systems, the NIC processor is much slower than the host processor. For example when host processing speeds of 450MHz are common today, the NIC processor speed is around 66MHz, and even 33 MHz [14]. As a consequence, the NIC implements the operations slower. Thus, division of labor between the NIC and the host should be balanced very carefully so that host processor cycles are saved by off-loading work to the NIC, but done so that overall latency does not increase because of the slower NIC processor

In some of the user level networking protocols, which are not implemented on top of smart NICs, this is not an issue because the NIC does not have a processor, and everything has to be taken care of by the host. However, this does not go along with the spirit of user level networking protocols.

2.1.3 Notification of Host and NIC

In user level networking protocols, since the transfer of message is handled by the NIC, the user application should be able to signal the NIC about the outgoing messages, and similarly the NIC should be able to signal the user application about incoming messages. In some cases where the NIC provides a hardware mechanism for notification, this can be done in hardware. However, this mechanism is still not present in many of the NICs and usually a software mechanism is used.

One way of implementing notification is to map some portion of the NIC memory to the user application. When a user wants to send a message, it writes a message to this memory location [11]. The NIC polls this memory location constantly to see if anything has been written. But, this mechanism places heavy burden on the NIC when there are many user processes communicating. With a single NIC, the NIC has to poll each of the user process slots to figure out if there is something to send. Considering the fact that the NIC has a slow processor and that it also has to handle incoming messages, there might be a significant degradation in communication performance when multiple processes are involved.

Another mechanism for implementing the notification requires going through the kernel. The NIC maintains a single queue for all outgoing messages. Since this is a centralized queue, processes should have protected access to this queue, and this can

be only provided by the kernel. In this case, we will have the cost of going through the kernel in our critical data path for communication.

2.2 A Survey of User Level Networking Protocols

In this section we briefly discuss about some important user level protocols and describe the design decisions made in their implementations.

2.2.1 Active Messages: AM

Active Messages(AM) [32] is from University of California, Berkeley, and it was designed for minimizing the communication cost and increasing computation and communication overlapping in large scale multiprocessors. On this framework, the message carries a pointer to user-level instructions that will take care of extracting the message from the network and integrating it with the ongoing computation. This is little different from an RPC call [5], since the goal of an RPC call is to do some computation on the remote node.

In AM, the communication is through endpoints which are created by the communicating processes. Each endpoint contains a send pool, a receive pool, handler table, a virtual memory segment, a translation table, and a tag. When a process wants to send a message it puts it in the send pool. Similarly messages that are received are stored in the receive pool for the receiving process. Incoming messages carry an index to the handler table that selects the handler to be invoked for that particular message. There are certain restriction on the operations the message handler can perform. Translation table stores names of possible destination endpoints and tags associated with them. A tag is a 64-bit integer that is that is used for associating messages and endpoints.

In AM, each request is matched with a reply. As a consequence each network operation takes a network round trip time. AM assume network to be free of errors, and to prevent messages to be dropped at the nodes it enforces a flow control mechanism. Each node has certain number of credits, and if a node runs out of credit, it should wait for the replies to obtain credit.

AM is an early work where user level processing of the incoming messages is done. The message is delivered to the user process by invoking the user level code immediately upon arrival of the message. Overhead for message processing is minimized, thus reducing latency, and increasing computation and communication overlap.

Although AM started as work in the context of large-scale multiprocessors, the work later has been extended and implemented for system area networks.

2.2.2 U-Net

U-Net [31] is one of the early works done on user level networking protocols. Its objectives were to provide low latency and increased network bandwidth even for small messages. Since it aimed to facilitate the development of new protocols by being able to implement them in the user space and not requiring a change in the kernel protocol stack. U-Net met these goals by using off-the-shelf hardware in the cluster environment.

In U-Net, communicating processes create endpoints through which they can access the network. Each endpoint contains message queues which hold descriptors of the messages that are to be sent or to be received. Specifically, there are three queues: a send queue, a receive queue, and a free queue.

Endpoints also contain communication segments which are basically memory regions. These memory regions are used as message buffers, or for storing message queues. Messages that are to be transmitted across the network are deposited into the buffers in the communication segment by the application. Similarly, the application provides buffers in the communication segment for the messages that it expects to receive.

Whenever an application wants to send a message, it prepares a descriptor with the destination, size and the location of the message. This descriptor is placed in the send queue. The NIC eventually sees this descriptor and puts the message defined by the descriptor into the network. When the NIC completes the send operation it informs the application by setting a flag in the descriptor.

When a message arrives at the NIC, it is copied to one of the buffers provided by the user in the free queue, and a descriptor pointing to the messages is pushed to the receive queue. The process can figure out about the message either by polling the receive queue, or it can block waiting for the arrival of the message. Very small messages can be delivered within the descriptor without the need to use a free buffer.

In U-Net, processes can access only to the endpoints created by that process. Also, the incoming messages are multiplexed to their respective destination processes depending on their tags. Thus, virtual protected access to the NIC is realized.

The original U-Net implementation provided zero-copy send/receive operations. It accomplished 32.5μsec one-way latency for single cell messages using ATM NICs between two 60 MHz SPARCstations. A true zero-copy implementation is later introduced in [33].

2.2.3 Illinois Fast Messages: FM

Illinois Fast Messages (FM) [26] is another user level networking protocol. The main objective in FM, similar to other ULNPs, is to optimize the software messaging layer that resides between the low level communication services and the hardware.

FM is implemented on top of Myrinet. FM provides reliable delivery. However, the order of messages in FM is not preserved. Also, FM does not provide a protective interface to the network. Thus, multitasking is not supported.

In FM, user can send a message from anywhere in its address space and no registration of user buffers is necessary. In fact, FM uses *Programmed I/O* (PIO) for transferring messages from user buffers into the NIC memory. In PIO, there is no need for address translation and no pinning is necessary. Thus, the need for address translation is eliminated.

There are four queues in FM. Send and receive queues reside in the NIC and receive and reject queues reside in the host. FM does no processing of messages in the NIC. The NIC blindly copies all incoming messages to the host.

FM implements a flow control mechanism in order to prevent buffer overflow. When a message is transmitted by the sender, it is kept in the sender's buffer. If the message is accepted by the destination, it is acknowledged, and the sender discards its own copy of the message. However, if the message is rejected by the destination, it is moved into the reject queue in the source and retransmitted later Multiple messages can be acknowledged by a single message or acknowledgements can be piggy-backed to outgoing messages as further optimization.

2.2.4 Virtual Memory Mapped Communication: VMMC

VMMC [13] is another user level protocol that provides protected and direct communication from virtual address space of the user process. On the receive side, VMMC provides zero copy, and no involvement of CPU is required. VMMC requires that the sender knows the buffer address on the receive side. However, when mapping a higher level communication layer to VMMC, an extra copy is required once the message arrives at its destination because the the receive side buffer addresses are not known by the higher layer. To address this problem, an extended version of VMMC, called VMMC-2, was proposed in [20]. VMMC-2 introduces the concept of transfer redirection to avoid extra copy on the receive side. The sender does not have to know the buffer address at the destination, and sends the message to a default buffer. If the receiver has already posted a buffer for the incoming message, the message is redirected and directly deposited to the provided buffer. If the receiver did not provide a buffer, then the message is copied to the default buffer. Later, it is delivered to its destination buffer.

VMMC-2 also provides a user level TLB to take care of virtual to physical address translation. In VMMC, address translation is done by interrupting the kernel. Interrupt to the kernel introduces a delay. By doing the translation in the user space the cost of context switching to the kernel is avoided.

2.2.5 GM

GM [6] is a commercial open source user level networking protocol from Myricom Corporation [3]. It runs on top of the Myrinet [14] hardware. GM is provided as a

low overhead communication abstraction to the Myrinet NICs. The design objectives of GM are low CPU overhead, portability, low latency, and high bandwidth.

GM provides a protected interface to the NIC, so that multiple user applications can share the NIC simultaneously. GM's protection depends on the memory protection mechanism provided by the hardware. GM provides reliable ordered delivery. Lost or corrupted packets are detected and retransmitted by GM.

As is the case with many ULNPs, user buffers should be registered with GM before they can be used for communication. These buffers are pinned down in the physical memory to enable DMA transfer in and out of these memory regions.

In order to establish a communication, user opens a port, and associates send and receive buffers with this port. User application may find out about the completion of the operations by polling the port's receive event queue. Receive event queue resides in the host's memory whereas send queue and receive buffer pool reside in the NIC's memory.

GM uses a "go back N" flow control mechanism. Whenever a message is lost, all the messages that has been sent after that message are resent. This wastes bandwidth but in the Myrinet environment the probability of a message getting lost is very low and Myrinet provides high bandwidth. So, a compromise has been made in favor of software simplicity rather than efficiency in the design.

2.2.6 VIA

The Virtual Interface Architecture (VIA) [7] is designed to provide high bandwidth, low latency communication support over a System Area Network. The VIA specification is designed to eliminate the system processing overhead associated with

the legacy network protocols by providing user applications a protected and directly accessible network interface called the *Virtual Interface* (VI).

Each VI is a communication endpoint. Two VI endpoints on different nodes can be logically connected to form a bidirectional point-to-point communication channel. A process can have multiple VIs. A *send queue* and a *receive queue* (also called as work queues) are associated with each VI. Applications post send and receive requests to these queues in the form of VIA descriptors. Each descriptor contains one *Control Segment* (CS) and zero or more *Data Segments* (DS) and possibly an *Address Segment* (AS). Each DS contains a user buffer virtual address. The AS contains a user buffer virtual address at the destination node. Immediate Data mode also exists where the immediate data is contained in the CS. Applications may check the completion status of their VIA descriptors via the *Status* field in the CS. A doorbell is associated with each work queue. Whenever an application posts a descriptor, it notifies the VIA provider by ringing the doorbell. In addition to the work queues, each VI can be associated with a completion queue. A completion queue merges the completion status of multiple work queues. Therefore, an application need not poll multiple work queues.

The VIA specification requires that the applications *register* the virtual memory to be used by VIA descriptors and user communication buffers. The intent of the memory registration is to give an opportunity to the VIA provider to pin down user virtual memory in physical memory so that the network interface can directly access user buffers. This eliminates the need for copying data between user buffers and intermediate kernel buffers typically used in the traditional network transports.

The VIA specifies two types of data transfer facilities: the traditional send/receive messaging model and the Remote Direct Memory Access (RDMA) model. In the send/receive model, there is a one-to-one correspondence between send descriptors on the sending side and receive descriptors on the receiving side. In the RDMA model, the initiator of the data transfer specifies the source and destination virtual addresses on the local and remote nodes, respectively. The RDMA write operation is a required feature of the VIA specification while the RDMA read operation is optional.

CHAPTER 3

MICRO-BENCHMARKS FOR EVALUATING ULNPS

In this chapter, we introduce a set of micro-benchmarks for evaluating the performance of ULNPs. First, we discuss why existing performance evaluation methods are not sufficient for giving an accurate picture of performance of the ULNPs as seen by the application. In the second section of this chapter, we introduce our micro-benchmark suite. Using the micro-benchmark suite, we evaluate the performance of four ULNPs in a comprehensive manner. The results of the evaluations are also presented.

3.1 Motivation behind Micro-Benchmarks

Traditionally, latency and bandwidth have been used for evaluating the performance of communication systems. In a standard latency test, the results are obtained by using a ping-pong test in which messages are exchanged between two nodes repeatedly. In this test, the same send and receive buffers at the end nodes are used in all the iterations of the experiment. This test gives a rough estimate of the latency of data transfer between two nodes. However, the situation corresponding to the standard latency test does not frequently happen in real applications. In most applications, different send and receive buffers are used for different send and receive operations.

As we explained in the previous chapter, in ULNPs sending/receiving from/to different buffers might introduce some additional overhead and affect the communication performance. Thus, the standard latency test is inadequate for bringing out this overhead in ULNPs.

Another example when the standard latency test does not give a good picture is when there are many connections in the system. In connection oriented ULNPs, information about each open connection needs to be maintained by the communicating entities. NIC resources such as memory and processing power are limited as compared to the host resources. If the information about each connection is kept on the NIC, it brings an additional overhead to the communication steps and the performance degradation might become highly noticeable. The standard latency test does not bring out the effect of such multiple open connections.

Bandwidth test is used to determine how much of the physical network's raw bandwidth is delivered by the low level communication protocol to the higher layer. Bandwidth test gives an upperbound on the achievable throughput by the ULNP. Similar to the latency test, the bandwidth test is inadequate for giving a complete picture of the performance that can be experienced by the application in realistic scenarios.

Some of the ULNPs provide different services that are not available in legacy protocols, such as sending/receiving from/to non-contiguous buffers, blocking, non-blocking sends/receives, completion queues in VIA. User may make use of these services or choose to implement them at higher layer. This decision process might be made easier if the user knows the performance of these services provided by the ULNP

layer. It is not possible to measure the performance of these services by the standard latency and bandwidth tests.

Neither bandwidth nor latency tests thoroughly evaluate the performance of ULNPs. There is a need for a micro-benchmarking suite that will reveal the strengths and weaknesses of ULNPs. This suite can be employed by users for evaluating and comparing different ULNP implementations. Then the user can choose the appropriate ULNP based on their needs. The benchmark suite can also be used to identify the bottlenecks in a given implementation. It can help in determining how well or bad the services provided to the upper layers by ULNP layer.

LogP parameters also give us a good performance indication of various components in the system. However, LogP parameters and the work presented in this thesis are rather orthogonal. LogP parameters provide more detailed information than core latency value. They indicate how much time is spent by the processor on receiving and sending a message, how much time is spent by the NIC, and how much time is spent in the network for data transfer. Also, the time between consecutive sends or receives are given as another parameter. However, LogP is measured under a special case, where there are only two nodes involved and the message is sent from the same buffer, and it assumes a scalable network. However as we discussed in Chapter 2, ULNPs behave differently under different conditions, and their performance changes.

3.2 Micro-Benchmark Suite

In this section, we describe our proposed micro-benchmark suite for evaluating user level networking protocols. For completeness, this suite includes the standard

latency test, bandwidth test, and measurement of LogP parameters. These are described as the first three micro-benchmarks. Five new micro-benchmarks proposed by us are described next.

3.2.1 Latency

Latency is defined as the time it takes for a message to travel from sender process' address space to the receiver process' address space. This time depending on the underlying communication layer, might consist of application buffer to system buffer transfer time, system or application buffer to NIC buffer transfer time, NIC to NIC transfer time, NIC to system or application transfer time, and system to application buffer transfer time. In protocols where zero copy on the sender side is implemented, application to system buffer time is zero, and similarly in protocols where zero copy on the receive side is implemented, system to application buffer transfer time is zero.

In our measurements, we measure latency as half of roundtrip time. There is a sender and a receiver. The sender sends a message and waits for an acknowledgement from the receiver. The receiver upon receiving the message from the sender sends a message of same size to the sender. This is repeated over many iterations and the average is reported as one way latency. This is similar to the standard ping-pong test used in the literature.

3.2.2 Bandwidth

Bandwidth reflects the communication bandwidth that can be achieved between two nodes. Similar to the latency test, there is a sender and a receiver. To measure the bandwidth, messages are sent out repeatedly from the sender node to the receiver node for a number of times and then the sender node waits for the last message to

be acknowledged. The time for sending these back to back messages is measured at the sender and the timer is stopped when the acknowledgement for the last message is received. Bandwidth is reported the total amount of bytes per unit time being delivered during the time measured. The number of messages being sent is kept large enough to make the time for transmission of the acknowledgment of the last message negligible in comparison with the total time.

3.2.3 LogP Parameters

For obtaining the LogP parameters, we use a set of experiments similar to those presented in [23]. The latency test, as mentioned in Section 3.2.1, gives us the quantity $(O_s + O_r + L)$, where O_s is the send overhead, O_r is the receive overhead and L is the time spent in the network. By using an experiment , we first obtain O_s, and then using another experiment we get O_r. Once O_s and O_r are found, L can easily be found by using the above equation.

To obtain O_s, an experiment similar to the bandwidth test can be used where messages are sent continuously from one node to the other. For small number of consecutive sends, the time to send each message gives us O_s. By increasing the number of consecutive messages, the pipeline between the sender and receiver becomes full, and the slowest stage determines the overall time. This corresponds to g. This is the same method used in [23].

The method used in [23] for measuring the receive overhead is similar to the latency test. However, in this test, a delay is introduced after sending the message and before receiving the acknowledgement. In the case that the delay is larger than

L, the measured one-way latency will become $O_s + delay + O_r$. Since we know O_s and $delay$, we can calculate O_r.

In the case that there is a trap to the operating system when a message arrives, the above experiment is not sufficient for measuring O_r. In this case, the arrival of message happens asynchronously, and is in fact included in $delay$. For measuring the receive overhead in implementations which employ interrupts, we propose the following modified experiment. A counter is implemented in a loop which runs for a certain amount of time on the receiver node before any message is sent to this node. Then a certain number of messages are sent to this node while the same loop is being executed on the receiver node. The difference between the value of the counter in both cases will reflect the time spent by the receiver host processor servicing the received messages. From this time the receive overhead for a single receive can be easily obtained.

3.2.4 Latency with Buffer Management

As described in Section 2.1.1, each time a new buffer is used in ULNPs, registration (pinning down for DMA transfer) of the buffer and/or, virtual-to-physical address translation for the buffer, might be necessary before the communication takes place. In ULNPs, in many cases, these are implemented without kernel support. Depending on the implementation, the buffer management can have adverse effect on the the communication performance.

In the latency test described earlier (Section 3.2.1, since the buffer is used throughout the test. Thus, the communication subsystem has a chance to cache the result of virtual-to-physical address translation. Thus, the effect of buffer management is

not visible in the latency result. We propose a new micro-benchmark to capture such effect.

In order to measure the overhead associated with buffer management in ULNPs, we modify the standard latency test. In the modified test, we allocate each buffer on a different page and each send operation is done from a different buffer. In this way, we make sure that the ULNP has to go through address translation phase (and pinning-down of buffer phase for true zero-copy systems) for each send. Thus, the cost associated with buffer management is added to results obtained by this micro-benchmark. By comparing these results to latency results, the cost of buffer management is obtained.

3.2.5 Latency with Multiple Connections

As we discussed earlier, one of the main objectives in ULNPs is to reduce the load on the host CPU so that it can be used for useful computation. In the case where smart NICs are used for communication, this might be accomplished by using the NIC processor and the NIC memory for some operations. However, one has to realize that, in most of the cases, the NIC processor is not as powerful as the host processor. Similarly, the NIC memory is not as large as the host memory. When there are many entities communicating over a single NIC, the load on the NIC might increase to the point where performance degradation becomes visible. We introduce a micro-benchmark to capture the scalability of a given ULNP in the presence of multiple connections.

For connection-oriented protocols, we propose to first establish a set of communications from a single node to other nodes. Then, we measure the latency experienced

between this node and one of the others in the presence of multiple open connections. In non-connection oriented systems this micro-benchmark cannot be used. However, there are some non-connection oriented systems, which establish communications internally (ie. not visible to the protocol user) between the communicating entities. These ULNPs maintain information about the open connections. The micro-benchmark can also be used in these kind of non-connection oriented systems to study the impact of multiple connections.

3.2.6 Latency with Blocking Completions

In most of the ULNPs, there are two mechanisms for detecting the completion of a send or receive operation. The user may poll for the completion or may block and wait for a signal to be informed about the completion of the event. Polling, although resulting in lower latencies, wastes CPU cycles. By blocking, the process yields the usage of CPU to another thread or process. However, in most of the cases the blocking mechanism is implemented using interrupts and it comes with a cost. This cost might be relatively high considering the low latencies achieved in ULNPs. We provide a micro-benchmark to measure the cost of blocking sends or receives.

In order to measure the effect of blocking for completions, we use a modified version of the latency test. The latency test polls for completion of send and receive operations. The modified test blocks for receive operations. Thus, the overhead of blocking is included in the results obtained by the micro-benchmark. We obtain the the cost of blocking by comparing the results of the micro-benchmark to the latency results.

3.2.7 Latency when sending from Non-Contiguous Data Segments

ULNPs provide services that might be useful for efficient implementation of to programming environment on top of them. One of these services is sending from non-contiguous buffers. In MPI, users can create non-contiguous data types and can send these data types to other nodes. Since legacy protocols do not provide mechanisms for sending from non-contiguous buffers, MPI layer has to concatenate the non-contiguous data into a single buffer. This means an extra copy of these buffers. If the ULNP layer supports flexibility for sending data from non-contiguous buffers, the extra copy can be avoided. With such flexibility, the ULNP layer can directly DMA out the data from these scattered buffers. Similar support can be provided at the receive side to receive data into non-contiguous buffers.

We propose a micro-benchmark to quantify the overhead introduced by the protocols when the user sends data from non-contiguous buffers. For the user of the protocol, this micro-benchmark helps in the decision process of whether or not to use the service. If the user can provide a more efficient implementation at a higher layer, sending from non-contiguous buffers may not be used by the user.

In the implementation of this micro-benchmark, we create a send descriptor with multiple data segments. These data segments are separated from each other in memory. These descriptors are then posted for communication. We obtain the latency for different number of data segments keeping the aggregate message size the same. This gives us the characteristic of behavior of the ULNP when multiple non-contiguous data segments are used in the communication.

3.2.8 Latency when using Completion Queues in VIA

Completion queues is a convenience provided by a particular ULNP, namely VIA. Since VIA is unifying the ULNPs under one standard, it is important to provide a micro-benchmark that measures the cost of services provided by VIA. Completion queues is a service that will provides flexibility to ease the higher layer programmer's job. A communicating process might have many endpoints for communication to many different endpoints. This is a common case in cluster environments. However, polling all of these nodes for incoming messages or for the status of outgoing messages can be costly. Completion queues are provided so that a process can associate all these endpoints with a completion queue. Then the process can check the completion queue for status of any send or receive completion. This mechanism can be very useful to a high level implementation, and in fact, used extensively by some MPI implementations (MVICH) [2].

We would like to determine the cost of using completion queues. This micro-benchmark can be employed to quantify the cost of a convenient service provided by the lower layer. It can help in the decision process of whether to use this facility or not.

To measure the cost associated with completion queues, we use a modified version of the latency test. In the modified version, we create a completion queue by using the primitives provided by VIA. Then we associate our send and receive queues with the created completion queue. During the communication, instead of checking the send/receive queues for completion, we check the completion queue. By comparing the results of this micro-benchmark to latency results, we obtain the cost associated with the usage of completion queues.

3.3 Results

In this section, we present our results of evaluating four ULNP implementations. First, we describe our testbed and the implementations we evaluated using our benchmarks. We evaluated three implementations of VIA, namely MVIA [2], Berkeley VIA [16], and CLAN VIA [1]. We also evaluated a GM implementation. In the second part, we present our results and give an analysis of the results.

3.3.1 Testbed

One of our experimental testbed consisted of 300 MHz Pentium II PCs with 128 MB of SDRAM, a 33 MHz/32-bit PCI bus and Linux 2.2 operating system. The PCs in the testbed were equipped with 33 MHz Myrinet LANAI 4.3 NICs and Packet Engines GNIC-II Gigabit Ethernet network interface cards. Myrinet and Gigabit Ethernet switches were used to construct two separate interconnection networks. The second testbed we used consisted of 450 MHz Pentium II PCs with 256 MB of SDRAM running the Windows NT operating system. These PCs were equipped with the Giganet CLAN 1000 NICs and interconnected through CLAN 5000 switches.

One of the VIA implementations we used in our test is MVIA [2]. MVIA is implemented as a part of the NERSC PC cluster project. In our tests, we used Release 1.0 which was made available in September, 1999. MVIA supports some Fast Ethernet NICs, and Packet Engines GNIC-I and GNIC-II Gigabit Ethernet NICs. If hardware support for doorbells is not provided in the NIC, MVIA uses a fast trap mechanism to trap to privileged mode, avoiding the high cost of a regular system call.

In our experiments we also used Release 2.2 of Berkeley VIA [16] for Linux 2.2.x kernels and Myrinet 4.x network interface cards. Berkeley VIA is implemented as a

part of the Millennium project, at the University of California, Berkeley. Berkeley VIA runs on top of Myrinet NIC's from Myricom. Some parts of the Berkeley VIA are implemented in the firmware of the NIC.

We also used CLAN VIA for Giganet. CLAN VIA implementation is provided by Giganet Corporation and runs on top of CLAN NIC, which is also provided by the same company [1]. CLAN VIA is the only implementation which takes advantage of the VIA hardware support provided by the NIC.

In addition to the VIA implementations, we also used GM [6] as another ULNP. We used 1.1.3 version of GM in our tests. GM implementation is provided by the manufacturers of Myrinet NICs. Some parts of GM are also implemented in the NIC firmware.

We present the results of our micro-benchmarks next.

3.3.2 Latency

One way latencies for all implementations for message sizes up to 32 Kilobytes are shown in Fig. 3.1. As can be seen from the figure, VIA on CLAN has the best latency. GM comes next. The other two VIA implementations have worse latencies - Berkeley VIA having a higher latency than MVIA. Latencies for small messages are shown in Fig. 3.2 . For a 4 byte message CLAN has a latency of about 9 microseconds. MVIA and GM provide latencies of around 22 microseconds, and Berkeley VIA has a latency of 38 microseconds.

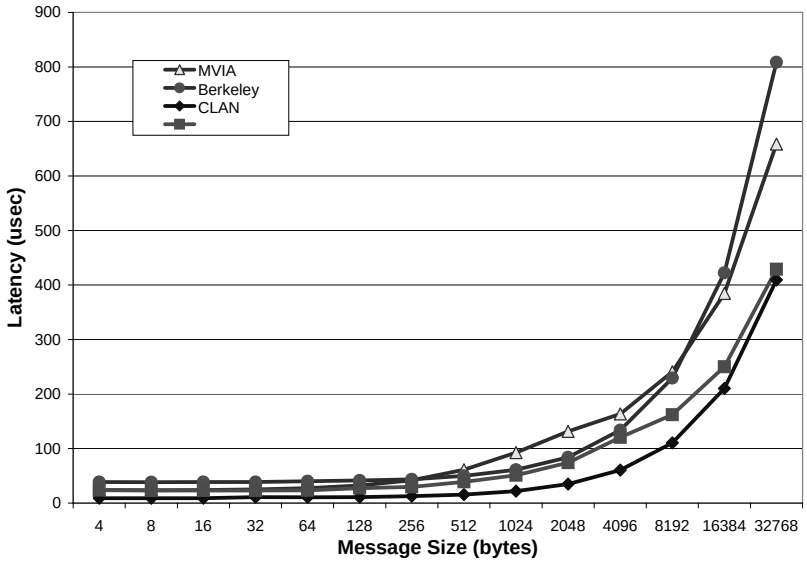

Figure 3.1: One way latency for different message sizes

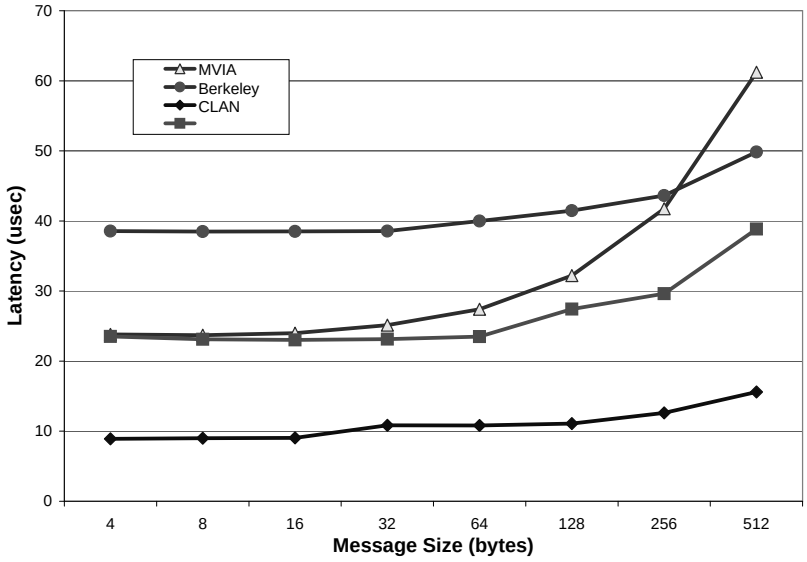

Figure 3.2: One way latency for small messages

3.3.3 Bandwidth

Bandwidth results are shown in Fig. 3.3. Except for large message sizes, CLAN provides higher throughput than other implementations. Highest throughput obtained by CLAN is around 640 Mbits/secs. Although, GM achieves half the bandwidth at a higher message size than other implementations (see Fig. 3.3), the bandwidth it achieves for higher message sizes is superior than other implementations. GM can go up to almost 700 Mbits/sec, which corresponds to approximately 70% of the physical channel capacity. For all message sizes MVIA's bandwidth is higher than Berkeley VIA. For 32 Kilobyte messages, a bandwidth of 560 Mbit/sec is observed for MVIA.

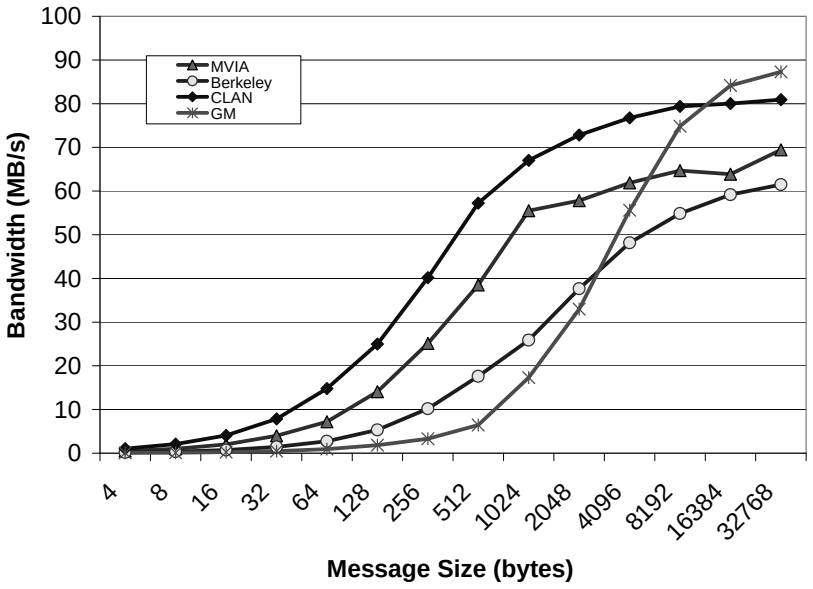

Figure 3.3: Bandwidth for different message sizes

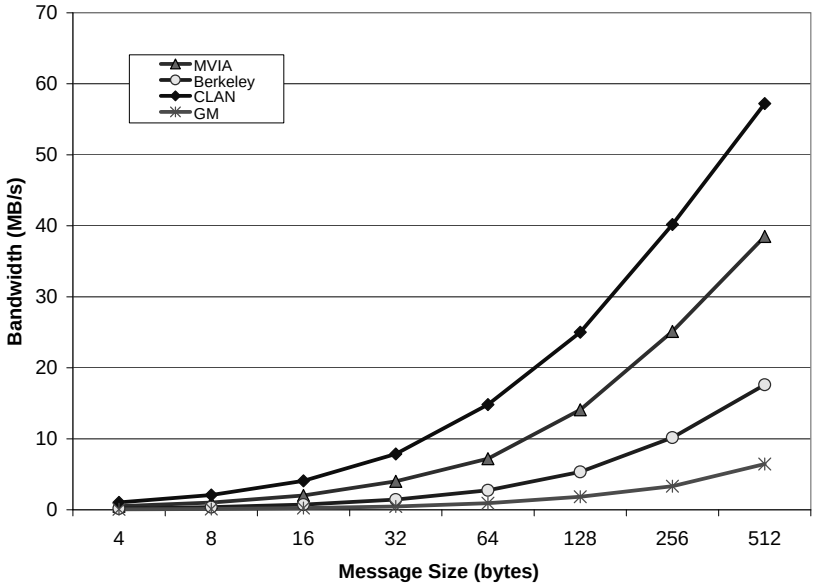

Figure 3.4: Bandwidth for small message sizes

3.3.4 LogP Parameters

The L, o, and g parameters of the LogP model for four different implementations of ULNPs are shown in Table 3.3.4. It should be noted that the L parameter in the LogP model is different from the latency measurement we introduced in Section 3.2.1. In fact, our latency measurement is the sum of O_s, O_r and L.

As we mentioned before, the LogP model can provide information about how much load sharing is done between the host and the NIC. The parameters O_s and O_r are the times spent by the host processor when sending and receiving a message, respectively. As can be seen from the results, MVIA has higher send and receive overheads. MVIA runs on Gigabit Ethernet NIC which is not a very smart NIC, Thus, all the work is done by the host processor. However, in Berkeley VIA, some of the work is done by the NIC processor, and this can be seen from the relatively high

L. GM is also another implementation in which some work is done on the NIC. Thus, it has a relatively high L. The relative contribution of each parameter to the overall latency can more easily be seen in Fig. 3.5.

(in μsecs)	MVIA	Berkeley VIA	CLAN VIA	GM
O_s	3.7	2.7	0.8	0.9
O_r	6.4	2.7	0.8	2.0
g	2.0	5.6	1.0	8.6
L	13.7	33.2	7.2	20.6

Table 3.1: LogP parameters for 4 byte messages

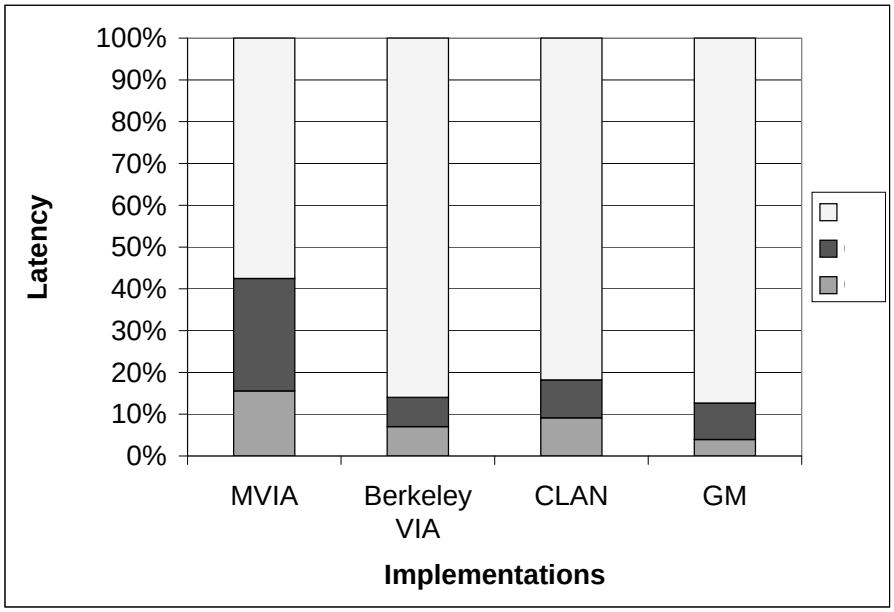

Figure 3.5: Contribution of each parameter to the latency in the LogP model

Figure 3.6 illustrates the send overhead (O_s) for different message sizes. The interesting result we observe is that MVIA is the only implementation in which O_s

increases with the increase in message size. In MVIA, host copies the message to the NIC. As the message size increases, the work that has to be done by the host also increases. This explains why O_s increases with message size in MVIA.

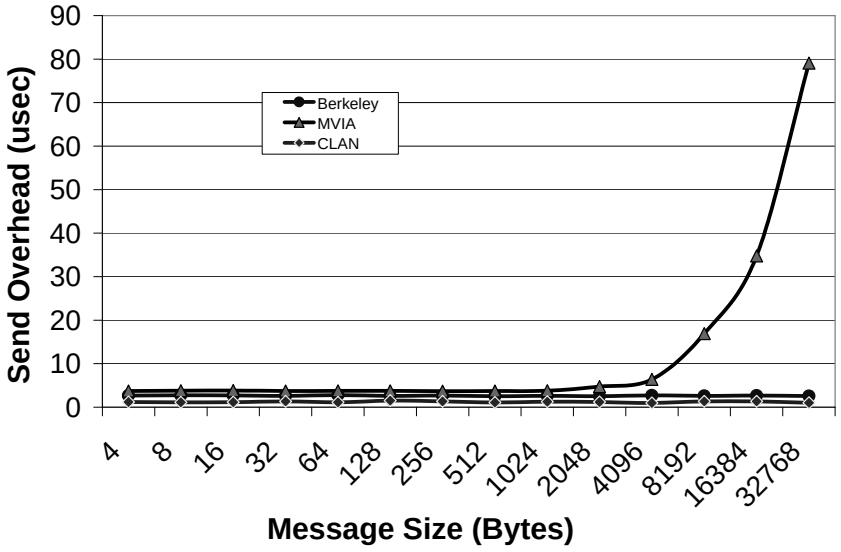

Figure 3.6: Send overhead for different message sizes

3.3.5 Latency with Buffer Management

In section 3.2.4, we stated that we would like to quantify the effect of buffer registration and address translation by using this benchmark. None of the ULNPs that we considered provide true zero-copying. As a consequence, automatic registration of buffers is not provided in these ULNPs. Thus, the results we provide in this section reflect only the cost of address translation done by the ULNP.

The differences between the latency results (Section 3.3.2 and the results obtained from this micro-benchmark are shown in Fig. 3.7. These differences reflect the cost of

address translation. It can be seen that changing the send and receive buffers has an effect on the latency of messages for GM and Berkeley VIA. The reason for this effect is that in Berkeley VIA the address translation tables are kept in the host memory and the NIC performs the translation. A software cache is used for caching the translations on the NIC. When only one buffer is used for say sending messages, after the first send, the required address translation entry is cached on the NIC memory and subsequent sends do not require the NIC to access the host memory. Since send and receive buffers in this micro-benchmark are used only once, the overhead of accessing the host memory for obtaining the address translation entries occur for each transfer. Depending on the application and the size and the type of software cache used by the NIC, applications may experience increased latencies due to buffer management. This micro-benchmark brings out such effects.

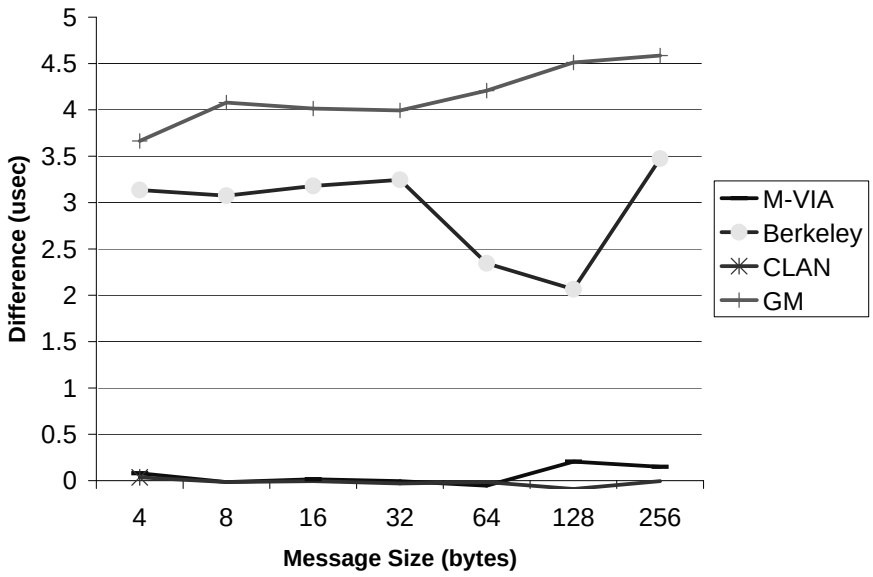

Figure 3.7: The cost of address translation for different message sizes

3.3.6 Latency with Multiple Connections

Figure 3.8 illustrates the latencies of short messages (4 words) when different number of active open connections exist in the the system. It can be seen that in Berkeley VIA, with increase in the number of connections, the roundtrip latency of messages increases significantly. Since Berkeley VIA does not use any hardware support for implementing doorbells, the firmware polls all the VIs for send descriptors to be processed. The increase in the number of VIs results in an increase in the polling time and therefore the message latency.

GM is not a connection oriented protocol. So it does not provide primitives for opening connections. However, internally GM maintains connections between communicating nodes. Using our micro-benchmark we created environments where a node establishes connections to multiple nodes, and tried to see the effect of these multiple connections in the performance of GM. Up to 16 open connections, we did not see any impact on the GM performance. We were not able to create more than 16 communicating nodes in our testbed. Thus, the results for GM are not included in Fig. 3.8. This micro-benchmark needs to be carried out in a large-scale system to study the impact of multiple connections on the performance of GM.

3.3.7 Latency with blocking completions

The cost of using blocking sends or receives is shown in Fig. 3.9. The data points are obtained by subtracting the base latency numbers (as indicated in Fig. 3.1 from latencies obtained when blocking completions used. It should be noted that GM does not provide blocking send primitives, but it does provide blocking receive operations. In our experiments these blocking receive operations were used. In GM, the cost of

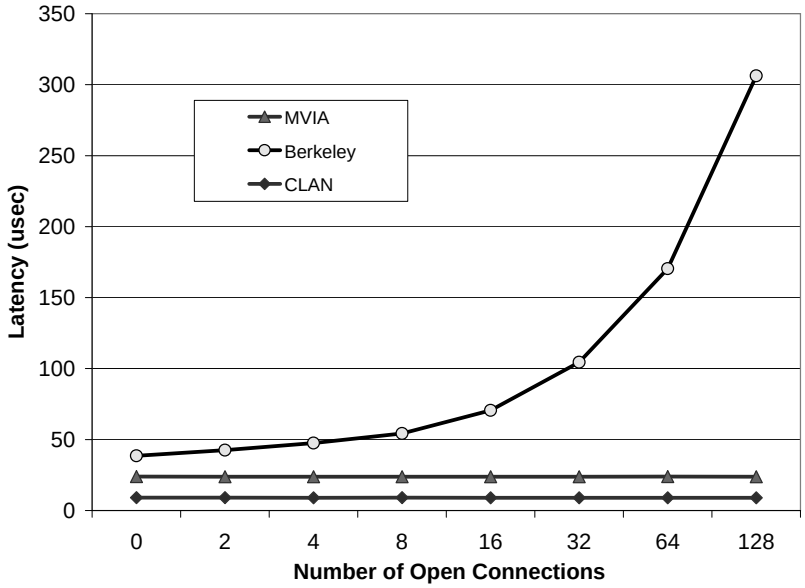

Figure 3.8: Latency observed in the presence of multiple open connections for 16 byte messages

blocking is observed to be quite higher. Thus, the user, depending on his/her need, might choose to poll rather than block for a completion of an operation. It is observed that , this cost seems to be close to zero in MVIA. In the MVIA implementation, before blocking for the completion of an operation, the status of the operation is checked repeatedly for a few times. If at this time the operation is found to be marked as complete, no blocking occurs. The other implementations block immediately without any polling. This shows why MVIA is superior to others in terms of lower blocking cost.

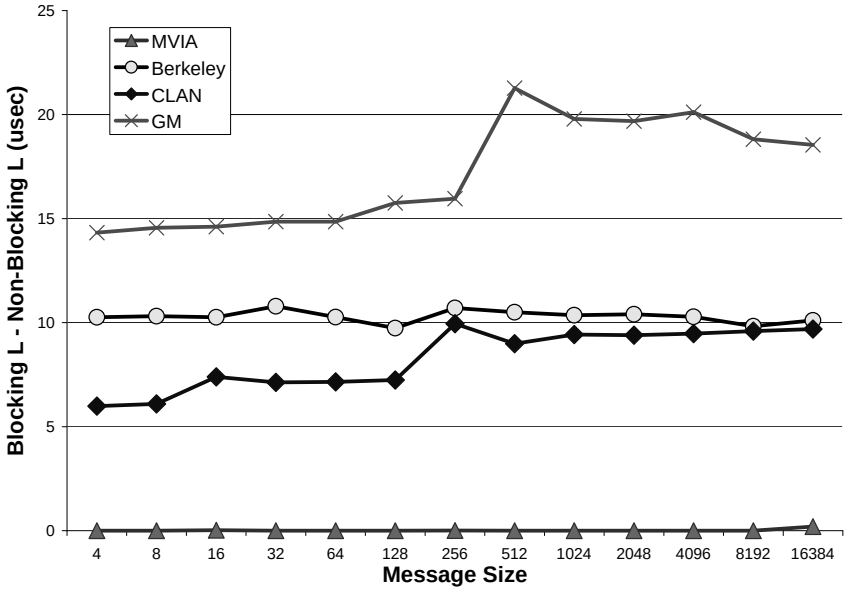

Figure 3.9: Cost of using Blocking Completions

3.3.8 Latency when sending from non-contiguous data segments

This benchmark is used to evaluate the performance of data transfers when non-contiguous data segments are used. GM does not provide this feature. While running our tests we found out that support for sending from or receiving to non-contiguous multiple buffers are not correctly implemented by these Berkeley VIA and MVIA. Thus, we were not able to evaluate these two implementations. This feature is in the VIA specification, however, these two implementations are at their initial stages and do not correctly implement these features yet.

Figure 3.10 shows the results for CLAN VIA. It can be observed that using many data segments can have significant impact on the latency. Especially if the data segments become smaller, the effect increases.

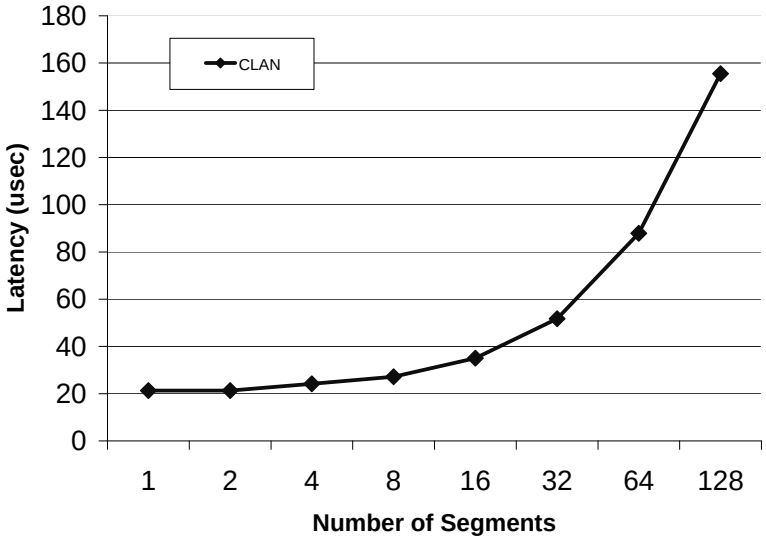

Figure 3.10: Latency when sending from non-contiguous buffers for 1024 byte messages

3.3.9 Latency when using Completion Queues

Completion queue mechanism is a feature of VIA as mentioned in section 3.2.8. So this micro-benchmark can not be used with GM. This micro-benchmark measures the latency of messages when their completions are checked through a completion queue. The number of work queues associated with a Completion Queue (CQ) is also varied such that its effect can be evaluated. The results from this micro benchmark is shown in Fig. 3.11. L_CQ indicates the latency when completion queue is used. L_Standard indicates the standard latency. It can be seen that using completion queues in Berkeley VIA leads to a 4 μsec penalty. With further investigation, we also realized that completion queues do not work correctly in Berkeley VIA and they do not return the correct results. Thus, this result corresponding to Berkeley VIA

is not completely stable. We do not see any effect of associating work queues with completion queues in MVIA. The increase in latency when a CQ is used for CLAN was observed to be only 0.8 μsec. It should be noted that the number of work queues associated with a CQ did not have any effect on the latency of messages for any of the three implementations.

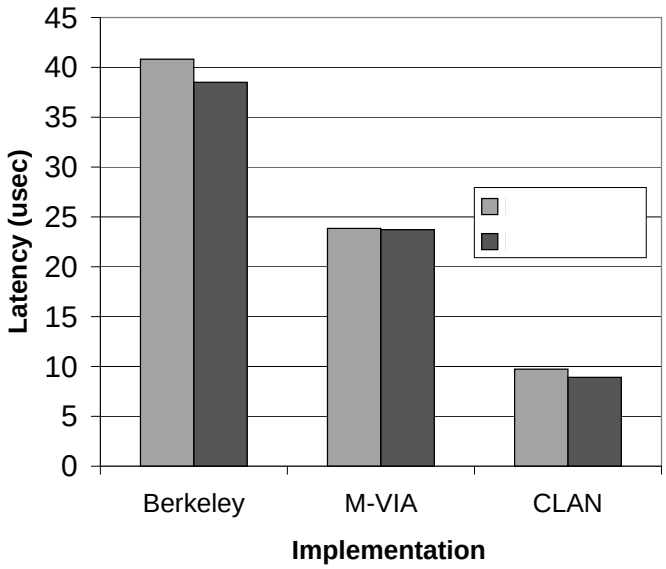

Figure 3.11: Cost of using completion queues in VIA

CHAPTER 4

MICRO-BENCHMARKS FOR EVALUATING USER LEVEL PROTOCOLS AT THE MPI LEVEL

For developing parallel applications, primitives provided by the ULNPs are not very well suited. Programming environment layer serves the purpose of providing a suitable environment for developing parallel programs and sits on top of ULNPs. *Message Passing Interface* (MPI) is one of the widely used programming environments for developing parallel programs. MPI is a standard and it has been adapted to a great extent. In this section, we provide the MPI level micro-benchmarks to further extend our evaluation of ULNPs to programming environment layer. By providing a set of micro-benchmarks at the MPI level, we see how the distinct properties of ULNPs are carried to MPI level. Also, we want to see how the performance delivered to the application is affected by this middle layer between the ULNPs and the application.

4.1 Motivations and Scope

In the last chapter, we used a set of micro-benchmarks to determine the performance of ULNPs. However, the performance delivered to the application goes through another layer. This layer, called the programming layer, resides between the ULNP

and the application, and affects the performance as seen by the application. To provide a thorough picture of the performance delivered by ULNPs to the application layer, one should also evaluate how the performance of ULNPs is passed through the programming layer. Since MPI is the commonly used programming environment for writing parallel programs in clusters, in this chapter we develop and present a set of micro-benchmarks at the MPI level. These micro-benchmarks extend the first set of micro-benchmarks. Along with the first set of micro-benchmarks, the MPI level micro-benchmarks allow us to determine how ULNP level bottlenecks can affect the performance at the MPI level.

Our goal is not to provide a comprehensive set of micro-benchmarks for evaluating a given MPI implementation, but rather come up with a subset that shows the effect of ULNPs at the MPI level.

Our benchmarks might be helpful to see whether performance bottleneck in a given system results from the MPI implementation, or from the ULNP. They might help MPI implementors to see how their implementations perform with respect to the ULNP implementations on which they are built on. From the user's perspective, the micro-benchmarks may be used to evaluate different MPI implementations and obtain insight about their strengths and weaknessness under different conditions.

4.2 Micro-Benchmarks

In this section we introduce our micro-benchmarks at the MPI level. These micro-benchmarks are similar to the previous set we introduced in the Chapter 3, but tailored for the MPI environment.

4.2.1 Latency

This micro-benchmark is used for measuring the latency between two communicating nodes. Latency is defined same as in section 3.2.1, and we measure it using the same methodology. However, in this case, we use MPI primitives for measuring latency. Again, latency gives us a good overall picture of the communication performance between two nodes. By using this micro-benchmark, we can also see how much extra overhead is added to the ULNP performance by the MPI layer.

4.2.2 Bandwidth

MPI-level bandwidth micro-benchmark is again similar to the ULNP level micro-benchmark introduced in section 3.2.2. It is measured the same way, but instead of using the ULNP primitives we use the MPI primitives. By comparing results at the MPI level and at the ULNP level, we can see how much of the ULNP level bandwidth is delivered to application layer.

4.2.3 Latency with Buffer Management

In most of the ULNPs, there are primitives for registering buffers before the communication begins so that they can be pinned down by the operating system for DMA transfers. However, at the MPI level, there is no such mechanism for registering the buffers used in the communication. In implementations of MPI on top of the ULNPs in which registration of buffers is required, this must be automatically handled by the MPI implementation. By using this benchmark, we would like to quantify the effect of the buffer management at the MPI layer. This benchmark is similar to the benchmark we introduced for buffer management in user level communication protocols in section 3.2.4. With this micro-benchmark, we can see how the effect of buffer

management at the lower layer is carried to the MPI level. In addition, we can see if there are additional overheads introduced by the MPI level for buffer management.

For measurements, we follow the same method that we used for ULNP level micro-benchmark. We modify the MPI level latency test such that each send is done from a different buffer. We measure the roundtrip time with the modified test, and compare the results to the MPI level latency test. The difference gives us the buffer management cost at the MPI level.

4.2.4 Latency with Multiple Connections

As we observed in Section3.3.6, some of the ULNPs have scalability problems. Their performance degrades in the presence of many connections . In cluster environments it is very common to see many nodes communicating at the same time. We might not also understand why the performance of a parallel application degrades when it is run on 64 nodes instead of 16. By using this micro-benchmark, we measure the communication performance seen in the environment where a node communicates with many other nodes. It should be noted that our aim is not to measure the effect of contention in the network, but rather how our communication subsystem behaves in the presence of multiple open connections.

To measure the latency in the presence of multiple connections we need to create multiple connections. Since MPI does not provide explicit primitives for establishing communications, we make sure that the connections are established by sending messages to different nodes at the beginning of our experiments. Then we carry out the latency test in the usual manner. The results compared with the latency test give us the effect of having multiple connections in the system.

4.2.5 Latency when using Non-Contiguous Buffers

Sending from non-contiguous buffers is a feature provided by MPI. In fact, this service is included in some of the ULNPs to support this MPI feature. Besides being a convenience for the programmer, sending from non-contiguous buffers might improve the performance by avoiding the extra copies required to put the message together. By using this micro-benchmark, we would like to see if the performance of this service (i.e. sending data from non-contiguous buffers) is good enough to be considered for use. In the case that the penalty is very large for using this service, we might as well choose to copy buffers into a single message at the application layer. Also, by using the micro-benchmark we can compare the performance of the service at the ULNP layer and the MPI layer. This comparison can show us whether the advantage of ULNP level non-contiguous send can be carried out to the MPI layer in a reasonable way.

In order to measure the effect of sending from non-contiguous buffers, we modify the latency test to send non-contiguous data. We use the mechanism provided by MPI to create a non-contiguous data type and send out data of this type. We carry out the ping-pong test with this type of data and measure the roundtrip times. The difference between results of the standard latency test and the results of this micro-benchmark gives us the cost of using non-contiguous buffers.

4.2.6 Effect of Completion Queues

Completion queues provide a convenient mechanism when dealing with multiple connections. By using the completion queues, a processor communicating with many nodes does not have to poll each endpoint to figure out about the status

of send/receive operations. (Section 3.2.8) MPI allows specifying any processor (MPI_ANY) in its receive primitive, which allows receving a message from any node. Completion queue is a very good way for implementing this receive operation. However, the question is how much overhead does this service incur. Again, our aim is to quantify the cost associated with the usage of this service, and see how it is related to the low level completion queue operations provided by the ULNPs.. For this micro-benchmark, we are limited to MPI over VIA implementations, since VIA is the only ULNP that provides this mechanism.

In order to measure the effect of completion queues we modify the latency test and in the receive primitive, we specify MPI_ANY instead of a specific node. Then we measure the roundtrip time with the modified receive primitive. The difference between the measured latency and the standard latency gives us the effect of using completion queues.

4.3 Results

In this section we present our results for evaluation of three different implementations of MPI. First we describe our testbed, and the versions of MPI used in our experiments. Next, we present the results. The results indicate that the MPI implementations over ULNPs deliver good performance to the application layer. However, the MPI implementations over ULNPs are not mature yet, and there is room for improvement.

4.3.1 Testbed

Our testbed consisted of a 16 node cluster, with Pentium II 300 MHz machines. The machines are two-way SMP machines, but in all of the measurements uniprocessor

kernels were used. The operating system used was Linux and the kernel version was 2.2.5.

We evaluated three different MPI implementations. MVICH [2] is from National Energy Research Scientific Computing Center (NERSC), and is implemented as a part of the NERSC PC cluster project. MVICH is implemented on top of MVIA and MVIA runs on top of Gigabit Ethernet in our environment. MVICH is in its very early stages, and has not been released yet. MVICH is available by sending an e-mail to via@nersc.gov.

Another implementation we evaluated is a ported version of MVICH on top of Berkeley VIA. Berkeley VIA runs on Myrinet hardware. Porting of MVICH on top of Berkeley VIA was done in our lab. It was a very quick port, and very little attention was paid to the performance issues. In theory, since VIA is a standard, there should not be any porting necessary for different VIA platforms. However, in practice, we found out this not to be the case. The port of MVICH from MVIA to Berkeley VIA required some effort. There are some minor differences in the interpretation of the standard by the two VIA implementations. Since Berkeley VIA is under development, its performance is not good as indicated in the last chapter. However, we chose to use it and our local port of MVICH on top of this in our experiments to demonstrate our micro-benchmarks.

The last implementation we used in our measurement is a port of MPICH for GM. MPICH [22] is from Argonne National Labs. A GM port is provided by Myri Corporation. It should be noted that, in our systems, we experienced a serious performance drop in the bandwidth test for GM for messages between 12 Kilobytes

and 16 Kilobytes. By changing the 3-way protocol[1] threshold to 12 Kilobyte in MPICH/GM implementation, we were able to get rid of this performance problem. In the original MPICH/GM implementation, the switch from eager protocol to the 3-way protocol occurs at 16 Kilobytes message size. For all the experiments in this thesis, we used this modified version of the MPICH/GM implementation.

Unfortunately, in our testbed we do not have a MPI over the CLAN VIA implementation. Thus, we could not evaluate the performance of MPI implementation on top of CLAN.

Next , we present our results of our micro-benchmarks.

4.3.2 Latency

One-way latencies for different platforms are shown in Fig. 4.1. As a reference, we also provide the performance MPICH over TCP/IP. For better illustration one-way latencies for small message sizes are shown in Fig. 4.2. Except for the case of quick MVICH/Berkeley VIA port, it is obvious that the performance of MVICH/MVIA and MPICH/GM are superior than the TCP/IP performance. It can be observed from the graph for small message sizes that one-way latency for MPI over TCP/IP is five times more than the latency for MPICH/GM or MVICH/MVIA. For larger messages, we observe the performance gap between the TCP/IP protocol and the ULNPs widen. This clearly justifies the use of ULNPs in cluster environments.

In Fig. 4.3, the raw MVIA latency is plotted along with the MVICH/MVIA latency to show the gap between the performance delivered at the ULNP layer and

[1]In MPICH implementation, for larger messages a 3-way protocol is used. In the 3-way protocol, the node sending the message first sends a notification telling the destination that it is about to send a message. The destination then replies back if it has sufficient resources for the incoming message. For smaller messages, eager protocol is used. In eager protocols, the sender sends the message immediately without asking the destination.

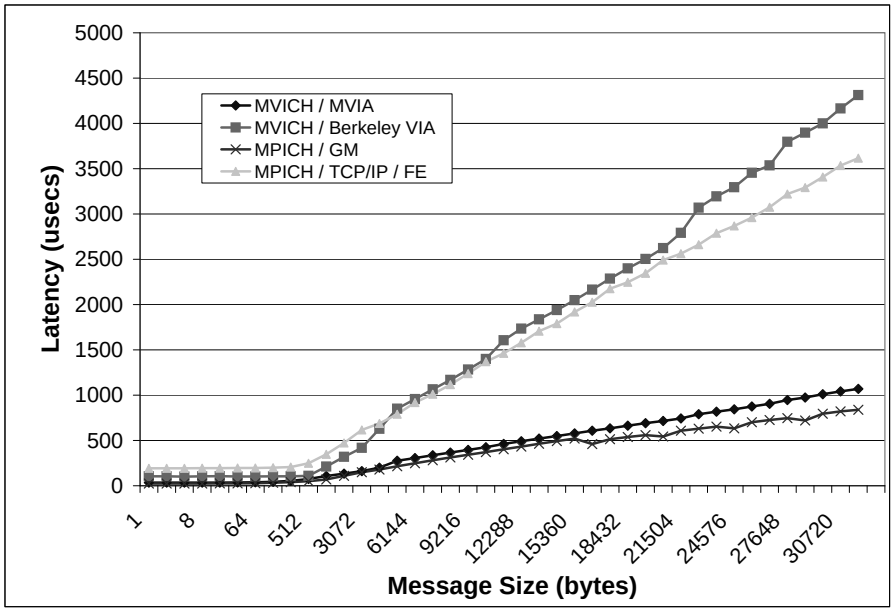

Figure 4.1: One way latencies for different MPI implementations for different message sizes

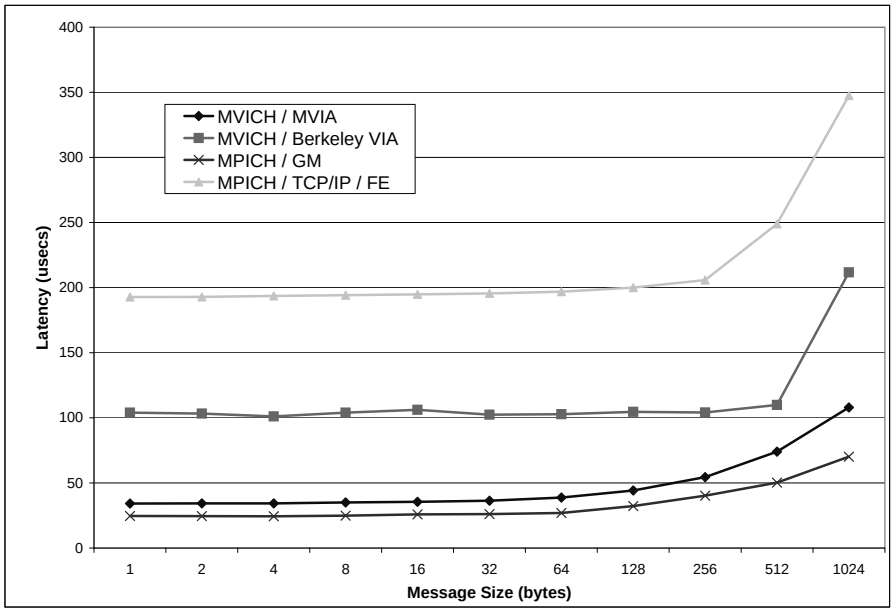

Figure 4.2: One way latencies for different MPI implementations for small message sizes

the performance delivered at the MPI layer. Up to 1 Kilobyte messages, MVICH implementation adds an overhead of around 11 microseconds. However, for larger messages the overhead increases with the message size. This indicates that an extra copy is being done for messages larger than 1 Kilobytes.

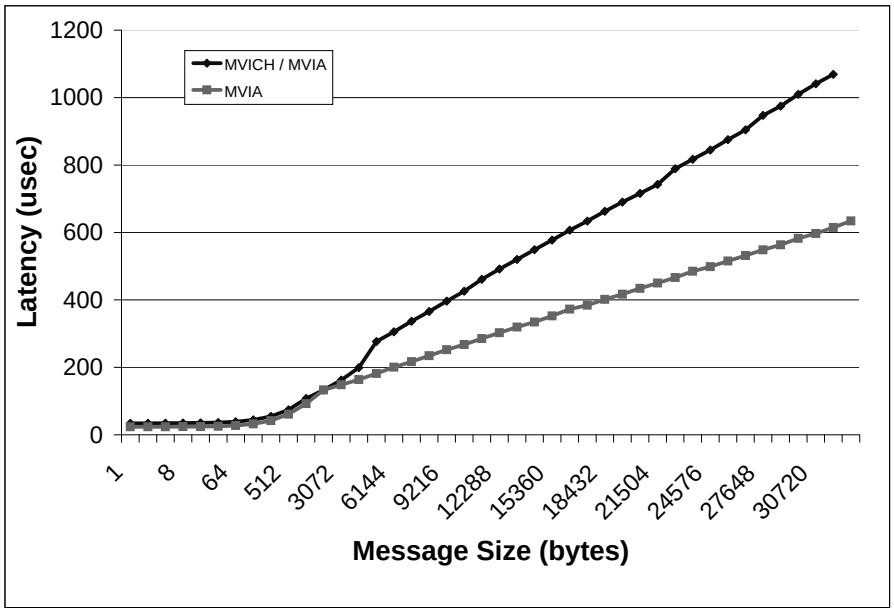

Figure 4.3: One way raw MVIA and MVICH/MVIA latency for different message sizes

Similarly, latencies for Berkeley VIA and our port of MVICH/Berkeley VIA are plotted in Fig. 4.4. It can be clearly seen from the graph that there are performance problems with our port, and the MPI layer adds a big overhead. MVICH heavily uses completion queues. As described in Section 3.3.9, completion queues do not work correctly in Berkeley VIA yet. During the porting, we provided a very simple and correct implementation of the completion queues. We think that this is the source of the performance problems.

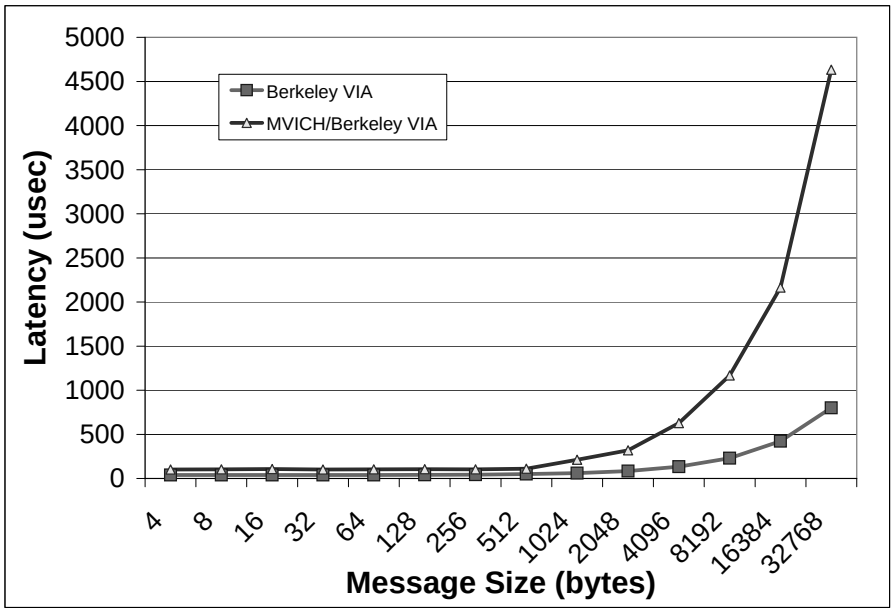

Figure 4.4: One way raw Berkeley VIA and MVICH/Berkeley VIA latency for different message sizes

The latencies for GM and MPICH/GM are plotted in Fig. 4.5. For short message sizes the MPI layer introduces as low as one microsecond overhead. As in the case of MVICH/MVIA, in the MPICH/GM implementation the overhead increases with message size. This tells us that there is an extra copy involved in GM for larger messages.

It can be observed that the MPI/GM implementation introduces the least overhead on top of the respective ULNP layer among the implementations we evaluated.

4.3.3 Bandwidth

The bandwidth delivered by the MPI implementations can be seen in Fig. 4.6 and in Fig. 4.7 for small messages. It can be observed that MPICH/GM delivers the

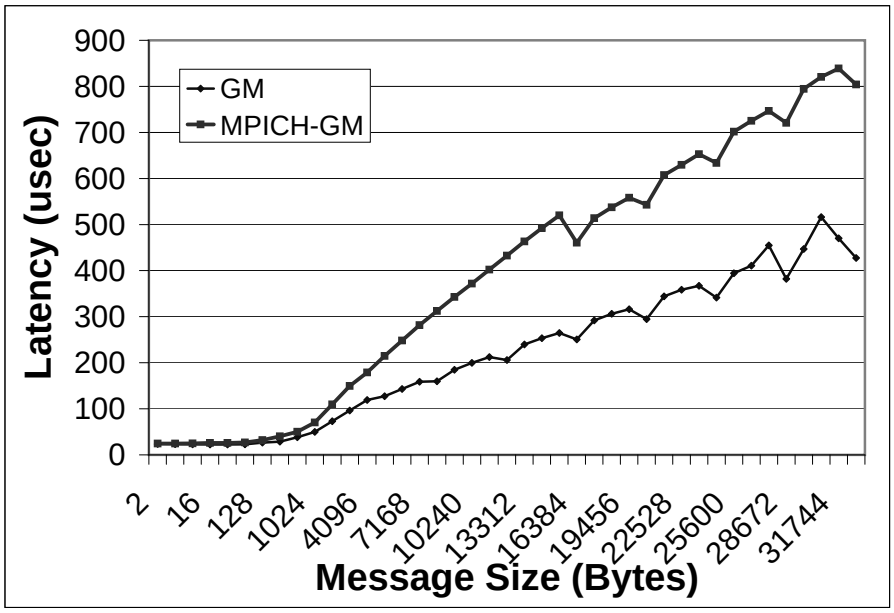

Figure 4.5: One way raw GM and MPICH/GM latency for different message sizes

most bandwidth to applications. However, for small messages MPICH/GM bandwidth is lower than MVICH/MVIA. Figures 4.8 - 4.10 compare the raw bandwidth delivered at the ULNP layers with the respective bandwidth delivered bMPICH/GM is able to deliver around 35% of the physical channel capacity and around 50% of the communication layer bandwidth to the application (see Fig. 4.10).

MVICH/MVIA delivers almost 40% of the ULNP layer bandwidth to the application layer (Fig. 4.8). In MVICH/Berkeley VIA implementation, we again experience performance problems and the delivered bandwidth is only around 10% of the ULNP layer's bandwidth (Fig. 4.9).

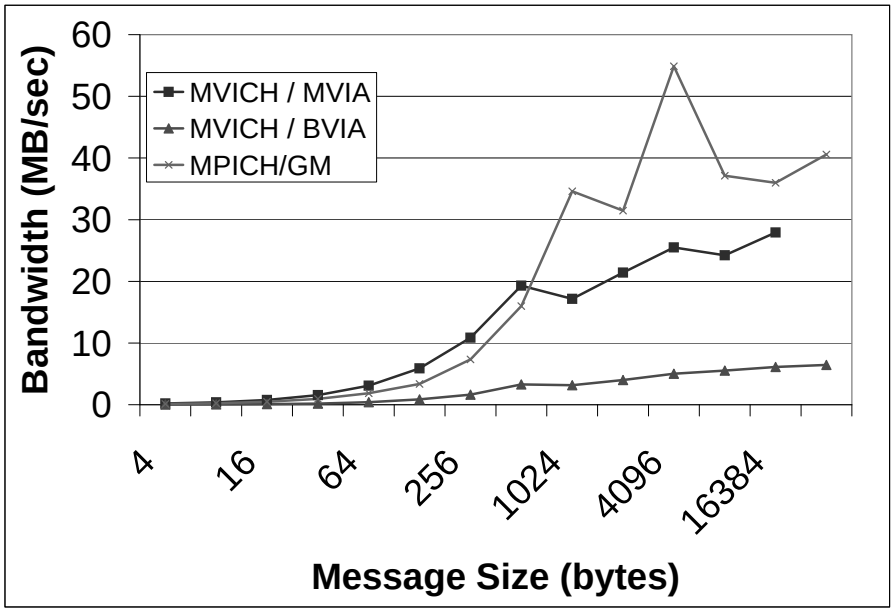

Figure 4.6: Bandwidth for different MPI implementations for different message sizes

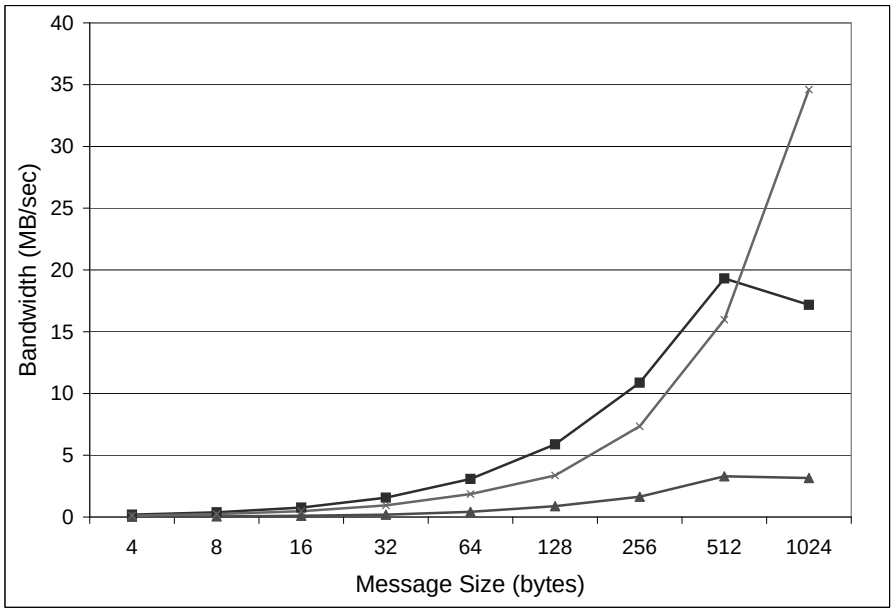

Figure 4.7: Bandwidth for different MPI implementations for small message sizes

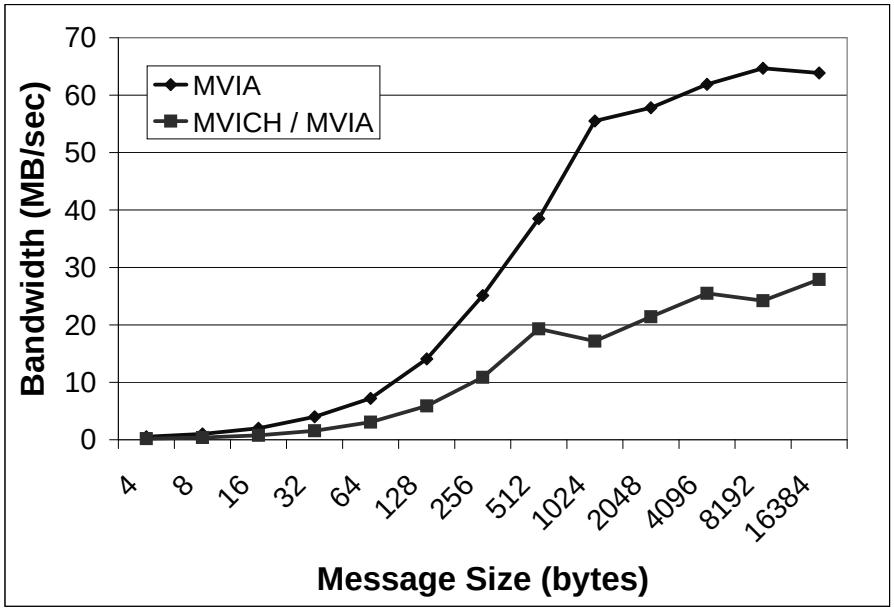

Figure 4.8: Bandwidth for MVIA and MVICH/MVIA for different message sizes

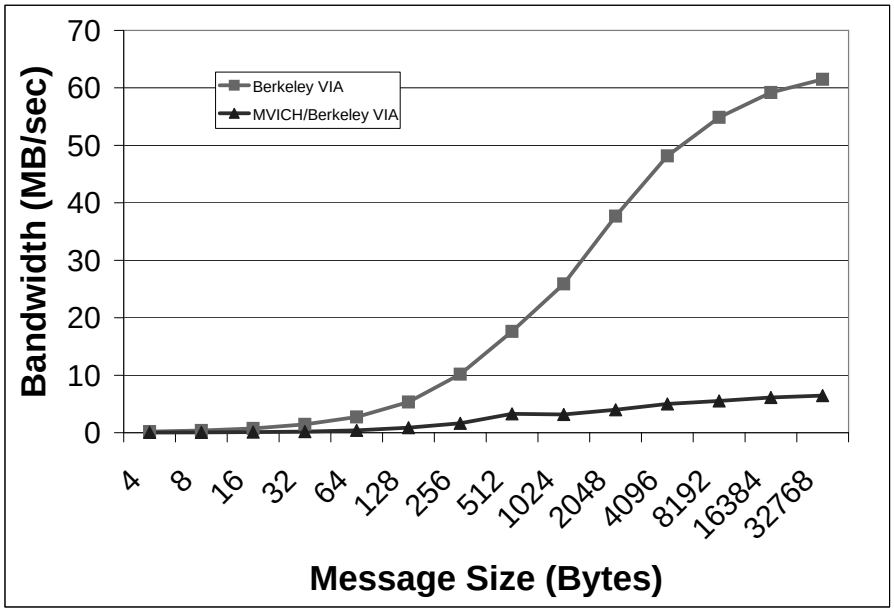

Figure 4.9: Bandwidth for Berkeley VIA and MVICH/Berkeley VIA for different message sizes

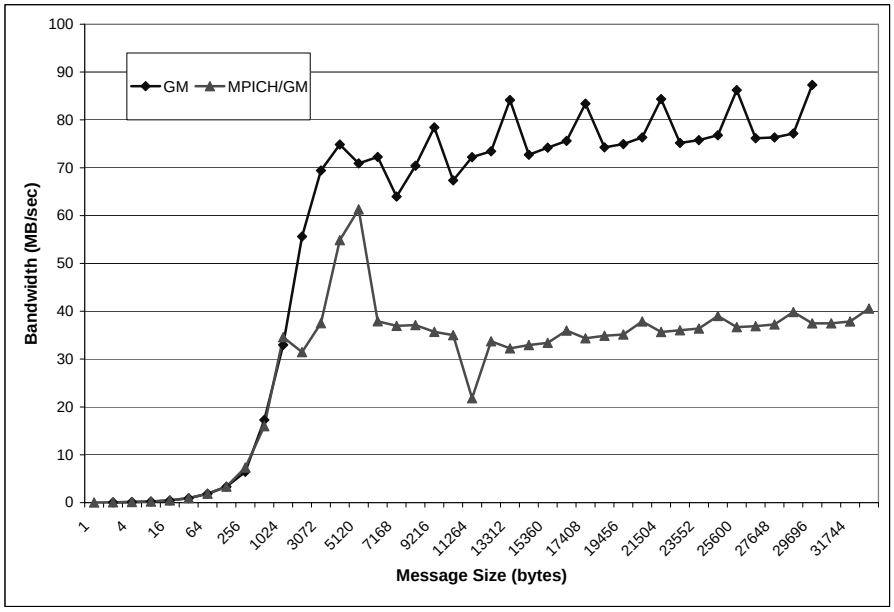

Figure 4.10: Bandwidth for GM and MPICH/GM for different message sizes

4.3.4 Latency with Buffer Management

Using this benchmark, we did not observe any overhead for buffer management in the data critical path for MPICH/GM, and MVICH/MVIA implementations. In Section 3.3.5, we observed an increase in latency for buffer management in GM. However, MPI implementation on top of GM does an extra copy and this extra copy covers the cost of buffer management. For MVICH/Berkeley VIA implementation, we observed that sending from different buffers introduces a constant overhead. In Fig. 4.11, latencies are shown for MVICH/Berkeley VIA when sends are done from different buffers.

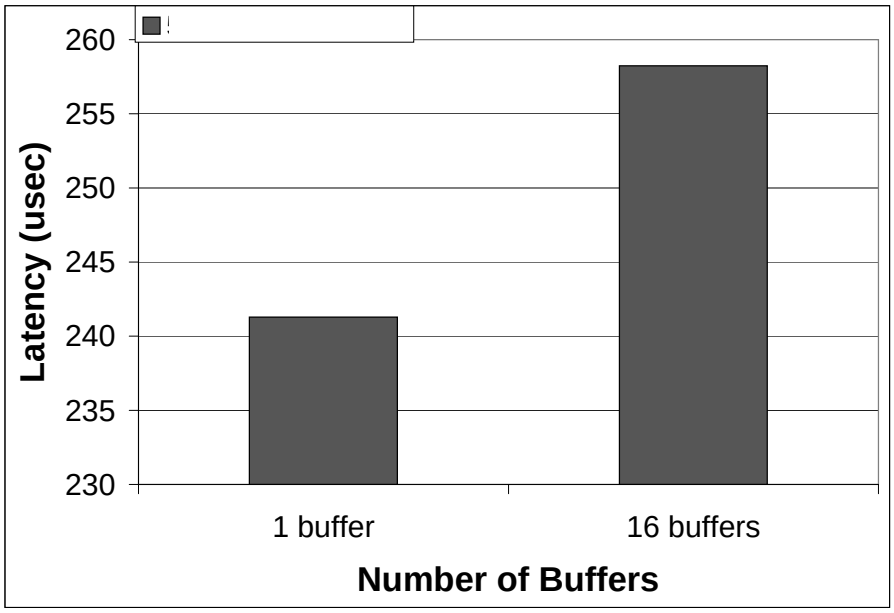

Figure 4.11: Cost of buffer management for 512 byte messages in MVICH/Berkeley VIA

4.3.5 Latency with Multiple Connections

From Section 3.3.6 we know that Berkeley VIA is the only implementation in which many open connections can affect the communication performance. At the MPI layer, our testbed permitted a maximum of 16 connections. We did not observe any performance degradation at the MPI layer for any of the implementations as the number of connections increased from 1 to 16. We project that with increase number of connections, the performance degradation for Berkeley VIA will be visible for 32 or more number of simultaneous open connections.

4.3.6 Latency when using Non-Contiguous Buffers

At the communication layer, CLAN is the only implementation we are able to evaluate with non-contiguous buffers benchmark. In the absence of MPI implementation over CLAN VIA, we are not able to provide results at the MPI level for CLAN implementation. MVIA and Berkeley VIA do not have this feature implemented. GM also does not provide mechanism for using non-contiguous buffers. However, MPI provides transfer from non-contiguous buffers. Regardless of the underlying communication layer, this service should be provided at the MPI layer. When running our benchmarks, we found out that MVICH implementation does not correctly implement sending from non-contiguous buffers. Thus, we were not able to run our micro-benchmark for MVICH/Berkeley VIA, and MVICH/MVIA implementations.

The results for non-contiguous buffer micro-benchmark for GM are shown in Fig. 4.12. MPICH/GM introduces significant overhead when sending from non-contiguous buffers for large message sizes. For different number of buffers the overhead stays the same. This suggests that MPICH over GM implementation copies the small segments to a buffer to assemble them.

4.3.7 Effect of Completion Queues

Since GM does not provide completion queues, this micro-benchmark is used for evaluating MVICH/MVIA and MVICH/Berkeley VIA implementations. Using our benchmarks, we did not observe any additional overhead when receving from any node (MPI_ANY). However, a close look at the MVICH implementation revealed that MVICH uses completion queues not only for receiving from any processor but for every completion detection. Thus, the cost of using completion queue is already

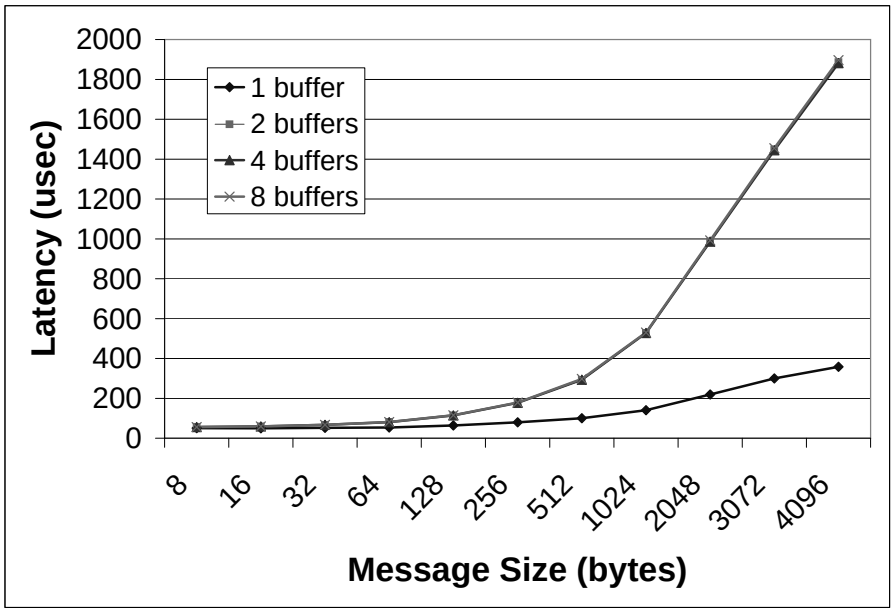

Figure 4.12: Latencies when sending from different number of non-contiguous buffers in MPICH/GM

added to each MPI send/receive operation. The fact that completion queues is in the critical path in all send/receive operation in MVICH makes efficient implementation of completion queues more desirable. In Section 3.3.9, we observe that MVIA does not introduce a significant overhead when using completion queues. Thus, in MVICH completion queues are used extensively without causing much performance degradation.

CHAPTER 5

WHAT THE APPLICATION SEES

To evaluate the performance of ULNPs we proposed a set of micro-benchmarks. We used our micro-benchmarks to evaluate the performance of different implementations of ULNPs, and MPI implementations on top of these. By using the micro-benchmarks, we tried to identify the bottlenecks, evaluate performance of different services, and see the effect of different design decisions. In this chapter, we use application level benchmarking Our aim is to see if the results we derived in the previous chapters for different user level networking protocols are consistent with the performance of the applications that are running on top of these systems.

5.1 NAS Benchmarks

We used *NAS Parallel Benchmarks* for our tests [4]. NAS parallel benchmark suite is provided by NASA Ames Research Center and it consists of eight applications derived from computational fluid dynamics problems. NAS parallel benchmarks are designed to objectively measure the performance of a parallel system. NAS benchmarks come in three different sets. The first set of benchmarks are "pen and paper" benchmarks and the implementation for a specific system is left to the vendors. Second set of benchmarks are MPI implementations of these "pen and paper" benchmarks by

NASA Ames Research Center. In our tests, we use this second set of benchmarks. Third set of benchmarks are serialized version of the parallel benchmarks for shared memory systems.

5.2 Results

Although NAS parallel benchmarks consists of eight applications we were not able to compile FT benchmark for any of the platforms on our systems. Thus, the results are given for only seven benchmarks.

We also profiled the communication characteristics of these benchmarks by using the profiling library that comes with MPICH. Knowing the communication characteristic for a given benchmark enabled us to interpret the results better. We derived the number and the size of the messages that are exchanged by a single node for of the benchmarks.

Completion times of the benchmarks in seconds are given in Table 5.2. The results are given for three user level networking protocols. Also, as a reference, we have included the results for MPI over TCP/IP implementation. All the platforms used Myrinet hardware for communication, except for MVIA which runs on Gigabit Ethernet.

We were not able get numbers for MG for both of the MVICH implementations, because these tests did not run to completion. MG uses collective communication operations. Unsuccessful results of these tests indicate a problem with the implementation of these operations in MVICH.

We observe similar trends to those that we observed at the MPI layer. In previous micro-benchmarks, GM and MPICH/GM provided better performance than

other communication systems. We see similar trends for the NAS parallel benchmarks. In all but one benchmark MPICH/GM performs better than the other two implementations. The exception is the LU benchmark. MPICH over GM performance is considerably worse for this benchmark. Profiling information revealed that LU is the benchmark which exchanges the most number of messages. Most of these messages are small. For example, one node sends around 31000 messages of size 620 bytes in LU benchmark. One explanation for the performance problem of GM might be GM's low bandwidth for small messages. In Section 4.3.3, we observed that GM has the lowest bandwidth for small messages. Also as indicated by the high g in LogP parameters for GM, it takes longer for GM than other ULNPs to pump out smaller messages to the network.

As is the case with the MPI layer results, MVICH/Berkeley VIA results are not not good. Although performance of Berkeley VIA is comparable to other ULNPs, MPI layer introduces high overheads. Thus, there is a need for better implementation of MPI on top of Berkeley VIA.

MVICH/MVIA performs reasonably well. There are some problems in the implementation of MVICH as indicated by the MG benchmark. MG benchmark does not run to completion in our testbed. However, it should be noted that MVICH is only version 0.03, and is at very early stages of its development. Development of MVIA is also going on and version 2.0 is released to be soon. Considering the immaturity of the implementations, the results are very promising.

Benchmark	MPICH/GM	MVICH/MVIA	MVICH/Berkeley VIA	MPICH/TCP/IP
BT	1047.06	1243.56	1471.18	1143.16
CG	19.93	24.32	35.98	37.73
EP	202.71	205.97	204.08	205.95
IS	7.84	10.02	22.89	30.36
LU	2274.1	828.55	1116.13	1095.32
MG	78.43	X	X	79.76
SP	804.08	873.23	1139.12	984.65

Table 5.1: NAS Benchmark Results: A Class Benchmarks on 4 processors. Total Completion Times

CHAPTER 6

CONCLUSION

In this thesis we have introduced a micro-benchmark suite to evaluate the performance of ULNPs. The suite includes a set of new micro-benchmarks designed exclusively for evaluating ULNPs, in addition to the standard performance evaluation benchmarks. The micro-benchmark suite can be used to assess the performance of a given ULNP in a bottom-up fashion. They are useful in identifying the bottlenecks, giving insights about the implementation, and analyzing the performance at an application level.

We used our micro-benchmarks to evaluate the performance of several ULNPs. Although many of the ULNPs we have evaluated are in their early stages of development, we found their performance to be very reasonable. In almost all of the cases, the ULNP performed better than the legacy TCP/IP protocol . This justifies the use of the new protocols for high performance computing in cluster environments.

We also observed that the implementations of ULNPs are not mature yet, and many problems exist with the implementations that need to be addressed. Generally, low level implementations delivered better results and MPI implementations were the source of bottlenecks and problems. There is clearly a need for better implementations

of MPI that utilize the potential of ULNPs. In particular, ULNPs achieve zero-copy but at the MPI level, we see that extra copies are introduced in the data critical path.

Because of the problems with some implementations, we were not able to fully test our micro-benchmarks both at ULNP level and MPI level. Not all the functionalities were implemented correctly in the ULNPs and the MPI implementations we evaluated. This is more visible at the MPI level and we were not able to obtain any results for some micro-benchmarks.

With the introduction of better implementations, and faster and smarter hardware for communication subsystems, the future for ULNPs looks very promising. In fact, there is some work being carried out for implementing traditional communication libraries on top of ULNPs [29]. Furthermore, some of the ideas that have emerged from ULNPs are being migrated into traditional protocols like TCP/IP [21]. More work is on the way for porting software distributed shared memory systems on top of ULNPs. [12]. Our micro-benchmarks can be employed to provide more insight to these implementations on top of ULNPs.

6.1 Future Work

In our micro-benchmarks, we used latency. Although latency as a metric, provides a good overall picture, in some cases measuring the LogP parameters can provide better insights. As we stated before, LogP parameters divide latency to its components and is a more refined metric than latency. In some cases, we might identify the component that affects the performance more accurately. For example, in section 3.3.6, we observed that the latency of Berkeley VIA increases with the increase in the number of open connections. By measuring the LogP parameters for different number of

connections we can investigate whether the host processor or the NIC is responsible for this additional overhead. Thus, LogP parameters and our micro-benchmarks can work in a complementary fashion providing better insights to the performance of a given ULNP.

In the previous chapters, we stated that we were not able employ some of our micro-benchmarks because fully functional implementations of the ULNPs and MPI implementations on top of these are not yet available. Once newer releases for these protocols are available, the tests should be run again to see the results of the micro-benchmarks. Also, by running all the micro-benchmarks, one can find out whether the performance problems are fixed in newer releases of these implementations or not. Thus, these micro-benchmarks can be developed as a tool for comparing and analyzing performance gains of different releases of communication layers.

BIBLIOGRAPHY

[1] GigaNet Corporations. http://www.giganet.com/.

[2] M-VIA: A High Performance Modular VIA for Linux. http://www.nersc.gov/research/FTG/via/.

[3] Myricom, Inc. http://www.myri.com/.

[4] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[5] RPC: Remote Procedure Call Protocol Specification Version 2 . http://www.ietf.org/rfc/rfc1831.txt.

[6] The GM API. http://www.myri.com/scs/.

[7] Virtual Interface Architecture Specification. http://www.viarch.org/.

[8] Fiber-distributed data interface (FDDI) - Token ring media access control(MAC). American National Standard for Information Systems ANSI X3.139-1987, July 1987.

[9] T. Anderson, D. Culler, and Dave Patterson. A Case for Networks of Workstations (NOW). *IEEE Micro*, pages 54–64, Feb 1995.

[10] ATM Forum. *ATM User-Network Interface Specification, Version 3.1*, September 1994.

[11] M. Banikazemi, B. Abali, and D. K. Panda. Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA). In *Proceedings of the CANPC workshop (held in conjunction with HPCA Conference)*, Jan. 2000.

[12] M. Banikazemi, D. K. Panda, and P. Sadayappan. Implementing TreadMarks on Virtual Interface Architecture (VIA): Design Issues and Alternatives. In *Proceeding of the Workshop on Scalable Shared Memory Multiprocessors(ISCA'2000)*, Jun 2000.

[13] M. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina. Virtual-Memory-Mapped Network Interfaces. In *IEEE Micro*, pages 21–28, Feb. 1995.

[14] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[15] J. Bruck et al. Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations. *Journal of Parallel and Distributed Computing*, pages 19–34, Jan 1997.

[16] P. Buonadonnaa, A. Geweke, and D.E. Culler. An Implementation and Analysis of the Virtual Interface Architecture. In *Proceedings of the Supercomputing (SC)* , pages 7–13, Nov. 1998.

[17] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 262–273, 1993.

[18] D. E. Culler, L. T. Liu, R. P. Martin, and C. Yoshikawa. Assessing Fast Network Interfaces. In *IEEE Micro*, Feb 1996.

[19] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware-Software Approach*. Morgan Kaufmann, March 1998.

[20] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient support for reliable, connection-oriented communication. In *Proceeding of the Hot Interconnects V*, Aug 1997.

[21] A. Gallatin, J. Chase, and K. Yocum. Trapaze/IP: TCP/IP at Near-Gigabit Speeds. In *Proceeding of the Usenix Technical Conference*, June 1999.

[22] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sep 1996.

[23] G. Iannello, M. Lauria, and S. Mercolino. Cross-platform Analysis of Fast Messages for Myrinet. In *Proceedings of the Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC '98)*, pages 217–231, 1998.

[24] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[25] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.

[26] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[27] Loc Prylli and Bernard Tourancheau. BIP: A New Protocol Designed for High Performance Networking on Myrinet. *In Proceedings of the International Parallel Processing Symposium Workshop on Personal Computer Based Networks of Workstations*, 1998. http://lhpca.univ-lyon1.fr/.

[28] M. W. Sachs and A. Varma. Fibre Channel. *IEEE Communications*, pages 40–49, Aug 1996.

[29] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual (VI) Architecture. In *Proceedings of the Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC '99)*, pages 91–107, Aug 1999.

[30] V. S. Sunderam. PVM: A Framework for Parallel and Distributed Computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

[31] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.

[32] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[33] M. Welsh, A. Basu, and T. von Eicken. Incorporating Memory Management into User-Level Network Interfaces. In *Proceedings of Hot Interconnects V*, Aug. 1997.

[34] H. Zhou and A. Geist. LPVM: A Step Towards Multithread PVM. Technical report, Oak Ridge National Laboratory, 1995.