TR-96-015

PM : A High-Performance Communication Library for Multi-user Parallel Environments

Hiroshi Tezuka Atsushi Hori Yutaka Ishikawa
E-mail: {tezuka,hori,ishikawa}@trc.rwcp.or.jp

Received 11 November 1996
Tsukuba Research Center, Real World Computing Partnership
Tsukuba Mitsui Building, 16th floor, 1-6-1 Takezono
Tsukuba-shi, Ibaraki 305, Japan

Abstract We have developed a communication library called PM for a workstation cluster using Sun SPARCstation 20/71's on a Myricom Myrinet. PM supports i) network context switching for the multi-user parallel processing environment and ii) FIFO message delivery. The *Modified ACK/NACK* flow control algorithm has been developed to realize these features. We implemented PM using several techniques for the Myrinet network interface such as *Immediate Sending* and obtained 24 micro seconds of latency and 32 M bytes per second of throughput with Myrinet 2.3.

1 Introduction

It has become possible to cluster workstations to make a high cost performance parallel machine, using high-performance Unix workstations and high-speed interconnect technologies such as ATM LAN, Fibre Channel and Myrinet[1]. In a workstation cluster, the SPMD programming model is employed in the sense that a single program runs on several

workstations. A process for the program on a workstation communicates with processes on the other workstations to exchange data. A window-based network protocol such as TCP is not suitable in such an environment because the increasing number of sender nodes needs more window buffers in a receiver node. If the total buffer size is limitted, larger number of sender nodes causes each window buffer size smaller and communication performance lower. Thus, the design and implementation of a scalable high performance network protocol is very crucial in a workstation cluster.

Recently, user memory mapped network drivers such as Active Messages[2] and Fast Messages[3] on Myrinet achieve a low latency and high-bandwidth communication. In these drivers, the user process directly accesses the network hardware so that kernel traps and data copys are eliminated. However, such a communication facility is only used by a single process because the process exclusively uses the network hardware resource.

We have designed an operating system called SCore and a communication library called PM using Myrinet to support multi-user parallel processing environment. In our workstation cluster, SCore is implemented as a daemon process called SCore-D on top of a Unix operating system, and it manages the user processes distributed to several workstations as shown in Figure 1. We call these user processes a *parallel process*. A parallel process directly accesses the Myrinet hardware resources like in AM and FM. However, using gang scheduling, SCore-D changes the context of the parallel process including the network state to allow multiple users to use the workstation cluster in a time space sharing (TSSS[4]) fashion. A SCore-D daemon process communicates with other SCore-D daemon processes via Myrinet to implement the gang scheduling. That is, two processes, the user and daemon processes, access the Myrinet hardware resources simultaneously.

PM supports i) Multiple communication *Channels* for a user and SCore-D daemon, ii) Channel context switching for multi-user parallel processing environment and iii) FIFO message delivery. To realize those functionalities, we have developed the *Modified ACK/NACK* flow control algorithm. We implemented PM using several techniques for the Myrinet network interface such as *Immediate Sending* and obtained 24 micro seconds of latency and 32 M bytes per second of throughput.

In this paper, we describe Myrinet briefly in section 1.1,

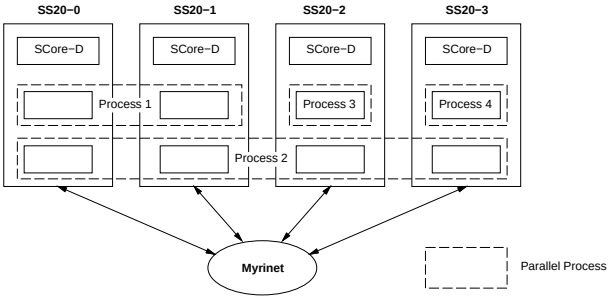

Figure 1: A parallel processing environment on a workstation cluster

then describe our design goal for PM in section 2. Section 3 describes the design and implementation of PM and section 4 shows a performance evaluation for PM. Section 6 describes related works, and section 6 is a summary of this paper.

1.1 Myrinet

Myrinet is a gigabit LAN commercially produced by Myricom Inc. using the research results of Mosaic[5] at Caltech and ATOMIC[6] at USC/ISI. Myrinet consists of three parts: 1)Link, 2)Host interface and 3)Switch as shown in figure 2.

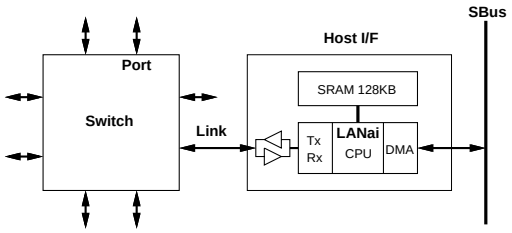

Figure 2: Myrinet

The Myrinet link is an 8-bit parallel, bi-directional data path which has a flow control mechanism using the STOP-GO method. The Myrinet host interface has a LANai chip which integrates a 16-bit microprocessor, a network interface and a SBus DMA controller. 128K bytes of high speed SRAM and line drivers are also on board. The LANai processor executes programs stored in SRAM to control the network interface and the DMA controller. Because all messages which are transmitted to and received from the link must be stored in this SRAM, a separate data transfer between the SRAM and main memory is needed. This

on-board SRAM is located in the SBus address space. By mapping this SRAM address area into the user address space, user processes can access data in the SRAM directly.

The Myrinet switch is a perfect permutational switch which employs cut-through routing. It adopts the source routing method, so the destination of each message is decided using routing information attached to the front of the message.

It is easy to implement new protocols on Myrinet because we can change the program on the on-board LANai processor. Accessing on-board SRAM directly from the user address space enables us to implement low latency message transfer using the polling technique. We need to take care in programming the LANai processor because the slow speed of the LANai processor (about 5MIPS) can become a bottleneck in performance [1].

Myrinet is distinguished from other LAN hardwares such as Ethernet by 1) High bandwidth: 80M bytes per second, 2) Guaranteed message delivery using hardware flow control, 3) Preservation of the order of messages transferred via the same path and 4) Host interfaces do not have addresses assgined and routing information must be supplied by software.

2 Design Goal

The workstation cluster environment which we have developed has the following characteristics: 1) It supports multiple users using gang scheduling, 2) A daemon process (SCore-D) and a user process use Myrinet simultaneously, 3) The number of connected nodes is not dynamically changed, 4) Configuration of all nodes are the same, and 5) The execution model is SPMD. From these characteristics of the workstation cluster, we established our design goals for PM as follows:

Support for multiple channels The PM *Channel* is a communication path which connects all nodes in the workstation cluster. All nodes communicate via a channel. A channel supports asynchronous communication. A message sent from a channel of the sender node is transferred to the same channel on the receiver node. PM does not support communication between different channels. PM must have multiple channels to allow

[1] Myricom has started shipping the second generation of Myrinet, which uses 32-bit RISC processors and achieves a 160M bytes/s link speed.

multiple processes (SCore-D and a user process) to use Myrinet simultaneously.

Low latency, high throughput, variable length messages Communication latency of PM should be as low as possible. And, to utilize the high bandwidth of Myrinet, PM should support a high throughput message transfer rate and high bandwidth communication. PM should support variable length messages from 8 bytes to around the physical page size to be used by inter-thread communications and I/O data transfers.

Guaranteeing message delivery PM should guarantee message delivery. As described in section 1.1, the hardware flow control feature in Myrinet gets round the problem of missing messages on the data link layer. But if a message is blocked by hardware flow control, all messages in other channels using the same link are also blocked [2]. To avoid this situation, messages should flow continuously, and a software flow control mechanism must be implemented to avoid missing messages during a receive buffer overflow.

Preserving message order PM should use a flow control method which preserves the message order.

Channel Context switching Each PM channel is occupied by a process because it polls the channel's data to receive messages. Because it is not possible to create as many channels as needed with the limited resources available from on-board SRAM, PM should support context switching of channels to allow a channel to be used by multiple processes in a time-sharing fashion.

Multicast assistance Myrinet hardware supports 1-to-1 message transfer only. To multicast messages to several nodes, the sender must send the same message to all receiver nodes. PM should decrease this multicast overhead by re-using the previously-sent message which remains in on-board SRAM.

3 Implementation

This section describes the implementation of PM needed to decrease latency, increase throughput, and flow control and context switching.

[2] For the first generation Myrinet product, if the link is blocked for longer than 50 miliseconds, the network is reset by hardware to avoid deadlock.

3.1 Decreasing latency

Protocol implementations which use system calls and interrupts and which are used in ordinary LANs such as Ethernet cannot satisfy the low latency requirement. Although Active Messages on the SPARCstation+ATM(SSAM)[7] uses special trap instruction to solve this problem, modifications to the operating system kernel are required.

We adopted a polling method to decrease the communication latency. In PM, the LANai processor transfers messages and the receiver thread of the user process polls the message arrival. Polling enables the receiver thread to know about a message arrival immediately and makes it possible to communicate with low latency[8]. Because polling by several processes simultaneously wastes CPU resources, PM also supports receiving messages using interrupts and this feature is used by SCore-D.

Further, PM uses the Immediate Sending techniques described in the next section to reduce the latency for large messages. PM also utilizes the fact that the communication library and the LANai program are also SPMD programs to achieve lower latency.

3.2 Increasing throughput

Because the Myrinet network interface can only access the data in on-board SRAM as described in section 1.1, the following procedures are needed to transfer data between main memory on each node.

1. Transferring the data by DMA from main memory to on-board SRAM.
2. Sending the data from on-board SRAM to the network.
3. Receiving the data from the network to on-board SRAM.
4. Transferring the data from on-board SRAM to main memory.

The sequential execution of DMA and message transmission cannot utilize the bandwidth of the DMA controller and the network interface. In this fashion, it is not possible to achieve high throughput. Although double buffering allows the DMA transfer of the next message to be executed at the same time that the current message is being sent can increase the throughput, it cannot lower the latency of each message transfer.

We developed an another technique called *Immediate Sending* that starts sending data from the SRAM to the

network immediately after the DMA transfer from main memory to SRAM begins. Immediate sending can both increase the throughput and lower the latency. Figure 3 shows the effect of Immediate Sending.

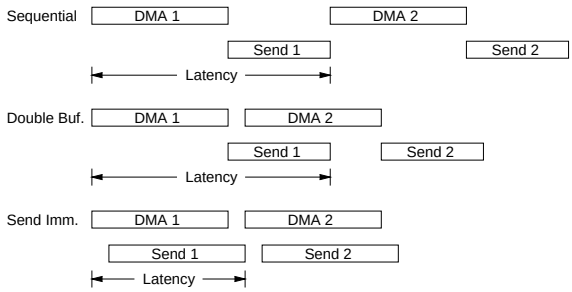

Figure 3: Immediate Sending

On a receiver node, it is not possible to execute receiving a message and DMA transfer simultaneously, because a CRC error is detected after the whole message is received. PM uses the double buffering technique on the receiver node to increase throughput.

3.3 Flow control

The flow control method for a workstation cluster should be scalable because it must work efficiently even if there are a large number of nodes. A window-based flow control algorithm is not scalable, because the receiver node must manage receive buffers dedicated to each sender node, and a larger number of nodes makes the effective buffer size smaller. Although the ordinary "ACK/NACK and re-transmit" method or "Return To Sender [3]" which sends back a messages which cannot be received does not have such a dividing buffer problem, they do not preserve the message order.

We developed a scalable flow control method which preserves the order of messages and implemented it in PM. This algorithm is called *Modified ACK/NACK* in the sense that it uses the ACK/NACK and re-transmit method and the sender/receiver state is introduced to guarantee the message order. Details of the Modified ACK/NACK algorithm are as follows. Here, $Msg(s, r, n)$ represents the nth message which sent from the sender node $Node(s)$ to the receiver node $Node(r)$. $Buf(r, n)$ represents the send buffer which corresponds to $Msg(s, r, n)$. $Ack(r, n)$ and $Nack(r, n)$ represents the positive and negative acknowledge for $Msg(s, r, n)$ respectively.

Sender node :

The sender node has two states: *normal* and *Not-to-send*.

Normal state :

- The sender node $Node(s)$ transmits a message $Msg(s,r,i)$ to the receiver node $Node(r)$. The send buffer $Buf(r,i)$ is not freed after the transmission.
- When $Node(s)$ receives positive acknowledge $Ack(r,i)$, it frees $Buf(r,j)$ where $j \leq i$.
- When $Node(s)$ receives negative acknowledge $Nack(r,i)$, it frees $Buf(r,j)$ where $j < k$ and re-transmits $Msg(s,r,l)$ where $k \leq l \leq i$. Then $Node(s)$ enters the *Not-to-send* state that inhibits transmission of messages except for these being re-transmitted.

Not-to-send :

- When $Node(s)$ receives $Ack(r,k)$, it frees $Buf(r,k)$ and enters the normal state to resume transmission.
- When $Node(s)$ receives $Nack(r,k)$, it re-transmits $Msg(s,r,l)$ where $k \leq l \leq i$ again.

Receiver node :

The receiver node has two states: *normal* and *Not-to-receive*.

Normal state :

- When the receiver node $Node(r)$ receives $Msg(s,r,i)$ normally, it returns $Ack(r,i)$ to the sender $Node(s)$.
- If $Node(r)$ receives adjacent messages $Msg(s,r,i) \sim Msg(s,r,k)$, it only returns $Ack(r,k)$ for the last message.
- If $Node(r)$ cannot receive $Msg(s,r,i)$ because of a receive buffer shortage, it discards the message and returns $Nack(r,i)$ to $Node(s)$. $Node(r)$ records $\langle s,i \rangle$ and it enters the *not-to-receive* state to rejects all $Msg(s,r,j)$ where $j \neq i$.

Not-to-receive state :

- $Node(r)$ does not receive $Msg(s,r,j)$ where $j \neq i$ and does not reply to sender.
- If $Node(r)$ receives the re-transmitted message $Msg(s,r,i)$, it returns $Ack(r,i)$ to $Node(s)$ and enters the normal state to resume receiving messages again.

Figure 4 shows an example of the flow control algorithm.

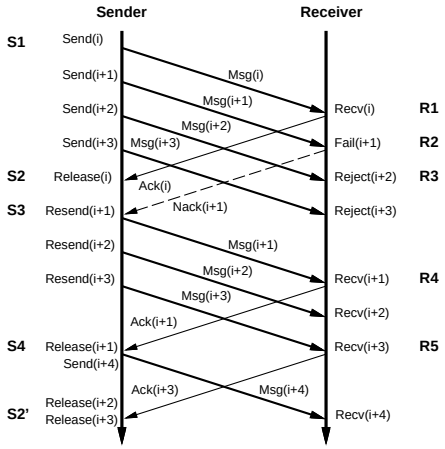

Figure 4: An example of Modified ACK/NACK flow control

In this flow control algorithm, the message order is preserved because succeeding messages are not received until the message which could not be received first is received normally. This algorithm requires a one-dimensional array to remember which message $\langle s, i \rangle$ cannot be received for each sender node. But this array does not occupy a large address space in the workstation cluster. The Myrinet, hardware which guarantees the delivery of messages, allows us to use this simple flow control algorithm. An advantage of Modified ACK/NACK flow control is that the sender node can know whether a message has arrived at the receiver node or not. We use this feature to implement the channel context switching described in the next section.

The drawback of this flow control algorithm is the possibility to increase the network load when re-transmission occurs, because the sender continues to transmit messages until a NACK arrives at the sender node.

3.4 Context Switch

The channel context of PM consists of data structures and buffers in the host main memory, and LANai on-board SRAM corresponding to the channel. To switch the channel context, these memory areas are saved to main memory and the next channel context is restored. Context switching of channels must not cause duplicate messages, missing messages or messages becoming mixed between contexts. The channel context should be switched while the channel

is in the *Stable* state, which means that no out-going messages or in-coming acknowledges are present. PM utilizes the ACK/NACK used in flow control to detect when the channel is in the stable state. When all ACK/NACK's for messages already sent are returned, the state of messages in the send buffer is confirmed as 1) Normally received, 2) Needs re-transmission or 3) Not sent yet, and it is confirmed that no messages are on the network.

Channel context switching in PM is performed as follows:

C1 Stop all sending on the channel except ACK/NACK.

C2 Wait until all out-going messages have been sent.

C3 Wait until all ACK/NACK to other nodes have been sent.

C4 Wait until all ACK/NACK's for sent messages have been received.

C5 *Synchronize all nodes.*

C6 Save current channel context to main memory.

C7 Restore next context from main memory.

C8 Resume sending.

In this procedure, synchronizing all nodes at *C5* must be done outside of PM, because PM itself has no mechanism to synchronize nodes. For this purpose, other PM channels can be used by an operationg system. Further, execution of the processes which share one channel must be controlled to be consistent with the channel context.

3.5 Work Load Assignment

To achieve high performance communication using Myrinet, the host processor should do as many tasks as possible and the LANai processor should do as few tasks as possible, because the LANai processor is slower than the host processor. In our PM implementation, the host processor takes charge of send buffer management and preparing send messages, and the LANai processor takes charge of sending and receiving messages, receive buffer management and flow control. In spite of these work load assignments, the flow control overhead executed by the LANai processor is large and it increases the communication latency.

3.6 Application Program Interface

Table 1 shows a part of the application program interface of PM. Because of the restriction of SBus that is required to change privileged registers to do DMA transfer to/from any address, PM uses a pre-allocated area for the send/receive buffers. Although this static allocation of buffers requires PM to get a send buffer before sending a message, and to return a receive buffer after use, copying data is avoided by constructing messages directly in the buffer area.

Table 1: API of PM (subset)

_pmLANaiInit	Initialize the host interface.
_pmInit	Initialize per process data.
_pmGetSendBuf	Get a send buffer.
_pmSend	Send a message.
_pmReceive	Receive a message.
_pmPutReceiveBuf	Return the receive buffer.
_pmSendActivate	Stop/Start sending.
_pmSendStable	Check if channel is stable.
_pmSaveChannel	Save a channel context
_pmRestoreChannel	Restore a channel context

4 Evaluation

We implemented PM on a 9-node SPARCstation20/71 cluster with Myrinet 2.3 as shown in Figure 5, and evaluated its latency, throughput, effect of multicast assistance and channel context switching.

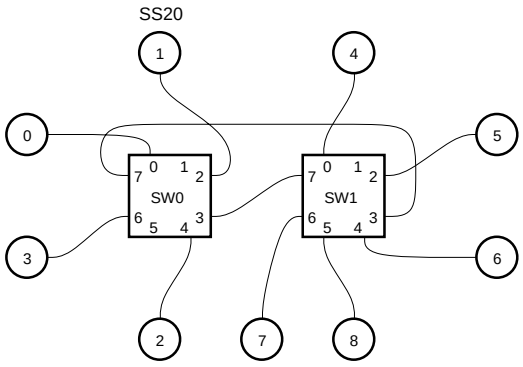

Figure 5: 9-node cluster

4.1 Latency

Figure 6 shows the one-way latency of PM. In this evaluation, we varied the message size from 8 to 4096 bytes, and measured 1) With Immediate Sending, 2) Without Immediate Sending.

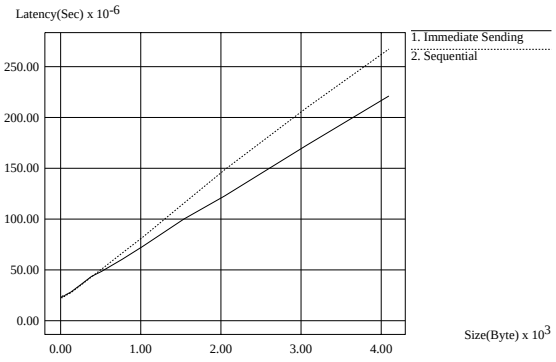

Figure 6: Latency

PM latency for an 8 byte message is about 24 micro seconds as shown in figure 6. Figure 6 also shows that the Immediate Sending technique decreases the latency for large messages.

4.2 Throughput

Figure 7 shows the throughput for PM. In this evaluation, we varied the message size from 8 to 4096 bytes, and measured 1) With Immediate Sending + receive double buffering, 2) With Immediate Sending only and 3) Without Immediate Sending and receive double buffering.

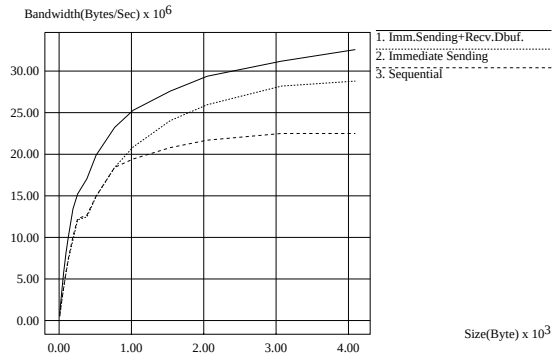

Figure 7: Throughput

Figure 7 shows that the Immediate Sending technique increases the throughput of large messages, and receive double buffering increases the throughput for all message sizes. The maximum throughput of PM is about 32M bytes per second with 4096 byte messages.

4.3 Multicast

Figure 8 shows the effect of multicast assistance on PM. In this evaluation, we varied the number of receiver nodes from 1 to 8, and measured 1) With multicast assistance, 2) Without multicast assistance: sender nodes repeat sending for all receiver nodes. Message size is 4096 bytes. Figure 8 shows the throughput per receiver node.

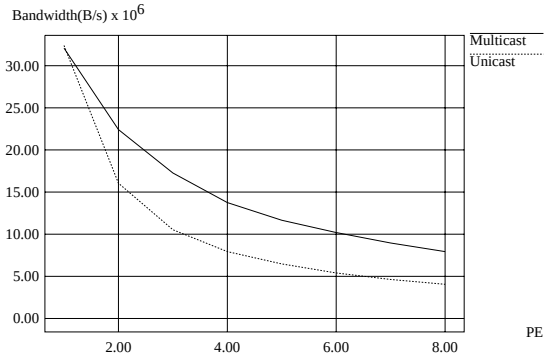

Figure 8: Multicast

Although repeat sending for N receiver nodes decreases the effective throughput to $1/N$, we achieved about twice the throughput of ordinary sending with 8 receiver nodes using multicast assistance in PM.

Due to the limitation of the Myrinet link speed, multicast to a larger number of receiver nodes requires another technique, for example copying and transferring data in a tree structure.

4.4 Context Switch

Table 2 shows the time to save and restore a channel context. We measured the four cases: 1) No message in the buffer, 2) Send buffer is full: 511 messages (12K bytes) are in the send buffer, 3) Receive buffer is full: 2730 messages (32K bytes) are in the receive buffer and 4) Both send and receive buffers are full.

Table 2: Context Switch time(mili seconds)

condition	save	restore
No message	0.13	0.11
Send buffer full	1.88	1.40
Receive buffer full	3.39	1.95
Send and Receive buffer full	5.15	3.22

It takes longer to save or restore the channel context when messages are in the buffers. This is because PM has a relatively large message buffer and the overhead to access the on-board SRAM via the SBus is large.

Although the amount of data to be transferred in saving and restoring the context is the same, saving the context takes longer than restoring it. This is because reading data from the SBus address space takes longer than writing data to it.

5 Related Works

There are several different implementations of the messaging layer on Myrinet, such as Myrinet API[9] (Myricom), Fast Messages on Myrinet[3] (Illinois University) and Active Messages[10] on Myrinet[2] (UCB).

Myrinet API supports multiple channels, scatter receiving and gather sending of messages and dynamic routing information generation. Myrinet API is optimized for throughput with large message, and its maximum bandwidth is about 26M bytes per second for 8192 byte messages. But the minimum latency of Myrinet API is large: about 100 micro second. Hence Myrinet API has no flow control mechanisms, reliable message delivery is made by an upper layer library called Mt. TCP/IP is also implemented on Myrinet API enabling Myrinet to be used as an ordinary LAN.

FM on Myrinet is designed to achieve high throughput with small message sizes, and its minimum latency is about 22 microseconds (8 byte messages) and maximum throughput is 17M bytes per second (1024 byte). MPICH[11], a portable implementation of MPI[12], on FM has a performance comparable with MPI on IBM SP2.

Figure 9 shows the latency and figure 10 shows the throughput for PM and FM for several message sizes.

As shown in figure 9, the latency on FM for a large

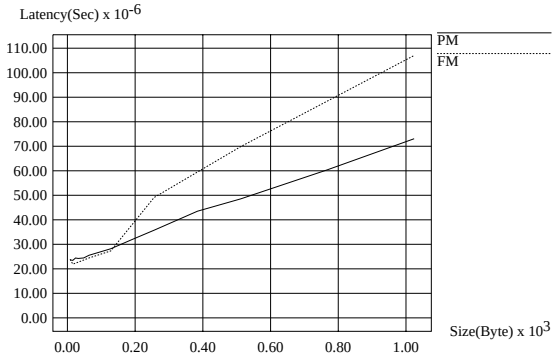

Figure 9: Latency of PM and FM

message size is larger than for PM. This is because 1) In FM, the message to be sent is copied to on-board SRAM by the host processor instead of the DMA, 2) PM adopts immediate sending as described in section 3.

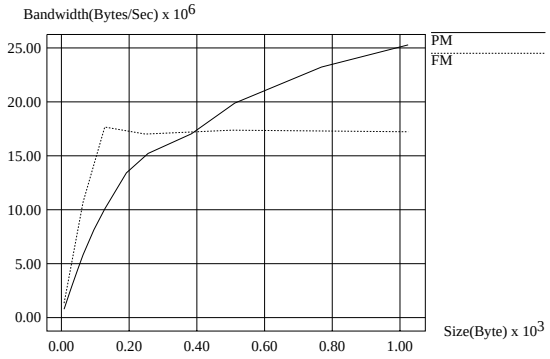

Figure 10: Throughput of PM and FM

[3] proposed $N_{\frac{1}{2}}$: the message size to obtain half the throughput at maximum bandwidth as an index of the throughput of a messaging layer. $N_{\frac{1}{2}}$ for FM is 54 bytes and it shows that the throughput with FM for short messages is very high. Otherwise, $N_{\frac{1}{2}}$ for PM is about 300 bytes. This is because the work load of the LANai is larger than for FM due to flow control and channel multiplexing. As shown in figure 10, the throughput for PM is larger than for FM for messages over 400 bytes.

FM uses Return To Sender (FM ver.1.0) or a window-based technique for flow control. The former does not guarantee the order of messages, and the latter has a problem with scalability.

The NOW (Network of Workstation) project at UCB

implemented Active Messages on Myrinet (LANai Active Messages). They achieved a 16.1 microseconds latency and a 28M bytes per second throughput. They constructed a Web search engine on a AM on Myrinet called Inktomi[13].

Myrinet API, FM and AM on Myrinet do not have a context switching mechanism like PM does, so it is difficult to use for any number of processes like our multi-user environment.

6 Conclusions and future work

We developed a communication library PM for a workstation cluster using Myrinet and SPARCstation 20's. PM supports not only low latency and high throughput communication, but also guarantees message delivery, preserves the message order, allows variable length messages, multiple channels and network context switching necessary in a multi-user parallel processing environment. We implemented PM using several techniques such as the Modified ACK/NACK flow control algorithm and Immediate Sending to support these features. We were careful in progra¡mming the LANai processor to achieve a high performance.

We are developing a 36-node SS20 cluster and implementing PM[14] on it. Our future works are to implement 1) a channel protection and 2) real-time capability to transfer continuous media data into PM.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su. "Myrinet – A Gigabit-per-Second Local-Area Network". *IEEE MICRO*, Vol. 15, No. 1, pp. 29–36, February 1995.

[2] http://now.cs.berkeley.edu/AM/lam_release.html.

[3] Scott Pakin, Mario Lauria and Andrew Chein. "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet". In *Proceedings of Supercomputing '95, San Diego, California*, 1995.

[4] Atsushi Hori, Takashi Yokota, Yutaka Ishikawa, Shuichi Sakai, Hiroki Konaka, Munenori Maeda, Takashi

Tomokiyo, Jörg Nolte, Hiroshi Matsuoka, Kazuaki Okamoto, and Hideo Hirono. Time Space Sharing Scheduling and Architectural Support. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, Vol. 949 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1995.

[5] C. L. Seitz, N. J. Boden, J. Seizovic, Wen-King Su. The design of the caltech mosaic c multicomputer. In *"Proceedings of the University of Washington Symposium on Integrated Systems"*, pp. 1–22. MIT Press, 1993.

[6] Felderman, R., DeSchon, A., Cohen, D., Finn, G.,. Atomic: A high speed local communication architecture. *Journal of High Speed Networks*, Vol. 3, No. 1, pp. 1–29, 1994.

[7] T. von Eicken, V. Avula, A. Basu and V. Buch. "Low-Latency Communication over ATM Networks using Active Messages". In *Proceedings of Hot Interconnects II, 1994 Palo Alto*, August 1994.

[8] E. A. Brewer, F. T. Chong, L T. Liu, J. Kubiatowicz and S. D. Sharma. "remote queues: Exposing network queues for atomicity and optimization". In *Proceedings of SPAA*, 1995.

[9] http://www.myri.com.

[10] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser. "Active messages: a Mechanism for Integrated Communication and Computation". In *Proc. of the 19th ISCA*, pp. 256–266, May 1992.

[11] http://www.mcs.anl.gov/home/lusk/mpich/index.html.

[12] http://www.mcs.anl.gov:80/mpi/.

[13] http://inktomi.berkeley.edu/.

[14] http://www.rwcp.or.jp/people/mpslab/score/Pm/.