Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication

Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori,
and Yutaka Ishikawa
Real World Computing Partnership
{tezuka, ocarroll, hori, ishikawa}@rwcp.or.jp

Abstract

The overhead of copying data through the central processor by a message passing protocol limits data transfer bandwidth. If the network interface directly transfers the user's memory to the network by issuing DMA, such data copies may be eliminated. Since the DMA facility accesses the physical memory address space, user virtual memory must be pinned down to a physical memory location before the message is sent or received. If each message transfer involves pin-down and release kernel primitives, message transfer bandwidth will decrease since those primitives are quite expensive. We propose a zero copy message transfer with a pin-down cache technique which reuses the pinned-down area to decrease the number of calls to pin-down and release primitives. The proposed facility has been implemented in the PM low-level communication library on our RWC PC Cluster II, consisting of 64 Pentium Pro 200 MHz CPUs connected by a Myricom Myrinet network, and running NetBSD. The PM achieves 108.8 MBytes/sec for a 100 % pin-down cache hit ratio and 78.7 MBytes/sec for all pin-down cache miss. The MPI library has been implemented on top of PM. According to the NAS Parallel benchmarks result, an application is still better performance in case that cache miss ratio is very high.

1. Introduction

We have been designing and developing a high performance network driver called PM on a Myricom Myrinet network which has a dedicated processor consisting of a DMA engine that can transfer data between the host memory and network. In order to eliminate the data copy between user and kernel memory spaces and to eliminate issuing operating system primitives, a user memory mapped communication facility has been realized[9]. In a user memory mapped communication, the communication buffer area, handled by the network hardware, is mapped to the user virtual address space. The network hardware is controlled by the user program instead of by the operating system. PM achieves not only high performance communication but also enables multiuser access in time sharing manner using gang-scheduling[9, 4, 5].

In PM 1.0, the communication buffer area is reserved as a special virtual address area, whose physical area is never paged out, so that the network hardware can trigger DMA. This area is called the *pinned-down* area. PM 1.0 does not allow the user program to pin a user virtual memory area to physical memory because the malicious user requests can exhaust physical memory. An application program will benefit from high performance communication if the program directly accesses the buffer area. However, an application program usually does not access the buffer directly. For example, a high-level communication library such as MPI may be implemented in a way such that a memory area specified in the user program must be copied to the PM buffer area. Though the PM 1.0 low-level communication library avoids data copy, the high-level communication library does not. Our experiments show that the MPI data transfer bandwidth is only 42 MBytes/sec while the PM data transfer bandwidth is 119 MBytes/sec on our *RWC PC Cluster II*, consisting of 64 Pentium Pro 200 MHz CPUs connected by a Myricom Myrinet network, and running NetBSD.

In this paper, a zero copy message transfer mechanism incorporated with the pinned-down area is designed in order to avoid memory copies between the user specified memory area and a communication buffer area. The API provides the pin-down and release operations for the application specified memory area and zero copy message transfer operations. Though the user is required to issue pin-down and release operations, pinned-down areas are controlled by the system. A pinned-down area is not freed until the total area pinned-down exceeds the some maximum size. If the total size of pinned-down areas exceeds the maximum size, some pinned-down areas are freed in the LRU fashion. Thus we call it the *pin-down cache*. The proposed mechanisms have

been implemented in PM 1.2. PM 1.2 is realized by a user-level library, a kernel driver, and a communication handler sitting on the Myrinet interface. No operating system kernel modifications have been done to implement the system except fixing some bugs.

The organization of this paper is as follows: In the following section, issues of low-level communication functions will be presented according to our experiments in PM 1.0. A zero copy message transfer protocol with the pin-down cache technique is proposed in sections 3 and 4. A new version of PM, PM 1.2 , based on the technique is described in section 5. PM 1.2 is evaluated in section 6. Section 7 describes related work and finally the paper is concluded in section 8.

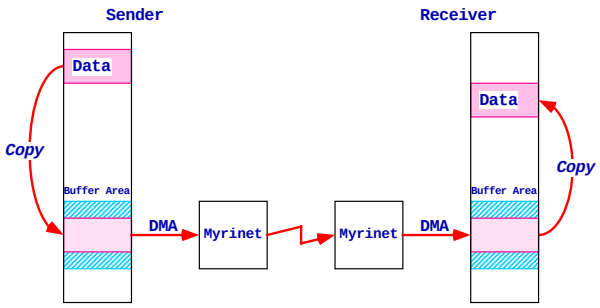

Figure 1. PM Data Transfer Flow

2. Background

Recent network hardware such as Myrinet has a dedicated processor with a DMA engine. The hardware enables the implementation of a low-level communication facility which then implements higher level communication facilities such as MPI. In this section, the requirements of a low-level communication facility are discussed according to experiments on PM 1.0 on the Myrinet network.

PM consists of a user-level library, a device driver, and a Myrinet communication handler which exists on the Myrinet network card. The Myrinet communication handler controls the Myrinet network hardware and realizes a network protocol. PM cooperates with an operating system to support low latency and high bandwidth in a multi-user environment[9]. Higher-level communication layers including MPI[8] and MPC++ MTTL (Multiple Threads Template Library)[6] have been implemented on top of PM. To achieve high performance communication, a user process directly accesses the Myrinet hardware and polls data without a system call or interrupt. The data flow between nodes in PM is depicted in Figure 1 and described as follows:

1) An application program stores data into the message buffer on the main memory at the sender node. Since the virtual address of the message buffer is mapped to the application program, the data is copied by the processor without issuing a kernel primitive.

2) The message buffer is transferred to the Myrinet SRAM area by the Myrinet DMA facility.

3) The message is sent to the receiver node by the Myrinet network.

4) The received message in the Myrinet SRAM area is transfered to the message buffer on the main memory by the Myrnet DMA facility at the receiver node.

5) Data in the message buffer is copied to the application specified area.

Since the DMA engine accesses a physical memory address, the message buffer area's physical memory must never be paged out. An operating system guarantees such an area in the form of a *pinned-down* area. PM 1.0 provides a buffer area as the pinned-down area so that the user program does not make a new pinned-down area for a communication buffer. The reason for not allowing the users to pin an area to a physical memory location is that the a malicious user's pin-down requests may exhaust physical memory. Providing the message buffer area has the following advantages: i) retransmission of a message is easily implemented at the congestion control level, and ii) asynchronous communication is implemented. The asynchronous communication capability at a low-level communication library is for to implementing higher level communication libraries. For example, in an MPI implementation, asynchronous communication is used not only for the MPI asynchronous message passing calls but also for passing control messages to implement other MPI features.

The PM 1.0 bandwidth on our *RWC PC Cluster II*, consisting of 64 Pentium Pro 200 MHz processors which have 512KB internal cache with the Myrinet LANai 4.1 PCI board, running NetBSD, is shown in Figure 2. The PM 1.0 bandwidth for an 8 KBytes message is 119 MBytes/sec excluding data copy to the application specified area while only 50 MBytes/sec bandwidth is achieved including data copy to the application specified area. This evaluation shows that for short messages, the overhead of copying data to the message buffer is not so crucial. But in for bulk data transfer, the copy overhead is dominant.

The overhead of CPU memory copy is further investigated in Table 1. In this evaluation, i) the `bcopy` C library function is used, ii) data larger than the external cache is transfered to eliminate a cache effect, and iii) several trials are done by sliding the copy destination address to examine

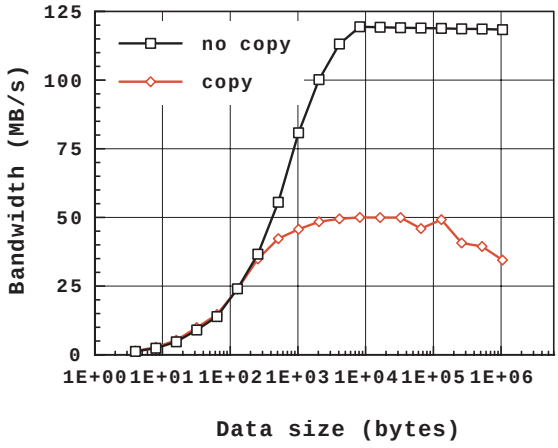

Figure 2. Data Transfer bandwidth in PM 1.0

the cache alignment effect. The result shows that memory copy bandwidth is heavily machine dependent.

If an Ultra Sparc 170 MHz is the target machine, the memory copy overhead is not crucial for bandwidth. But copying data yields communication latency. Thus, copy elimination at the low-level communication level is needed to achieve high performance communication for any machine.

Supporting an asynchronous communication with a zero copy message transfer mechanism is required at the low-level communication level. However, since asynchronous communication requires a buffer area, an extra copy between a buffer area and user specified area cannot be avoided. Thus, the new PM version, PM 1.2, supports both asynchronous communication and zero copy message transfer primitives. The zero copy message transfer primitive is based on remote memory write. A higher-level communication library may combine these primitives to implement the library features.

Table 1. Data Copy Bandwidth (MBytes/s)

Machine	minimum	maximum
SS20/71 (75 MHz)	13.2	25.8
Ultra SPARC (170 MHz)	188.0	205.5
Alpha PC (433 MHz)	79.6	124.4
Pentium (166MHz)	31.0	31.5
Pentium Pro (200MHz)	52.3	48.7

3. Zero Copy Message Transfer

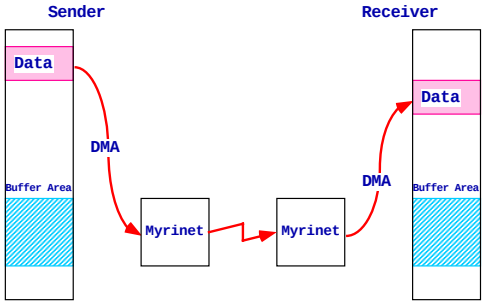

Figure 3. Zero Copy Message Transfer

As shown in Figure 3, a zero copy message transfer is realized by the Myrinet DMA capability. The user program accesses only virtual addresses while the DMA engine only accesses physical memory addresses. To realize a DMA-issued message transfer, both sender and receiver physical memory addresses must be known and must be pinned to physical memory. Two designs are considered:

1. Message Passing Rendezvous
   When an application program issues the synchronous message receive primitive, the receiver receives the message and stores data to the destination area specified by the message receive primitive. In this implementation, the sender does not need to know the remote memory address where the data is stored.

2. Remote Memory Write
   The sender sends a message to the specific destination address on the receiver so that the receiver receives the message and stores data to the destination address.

We choose the latter design for the following reason: Message passing rendezvous is only effective if it is guaranteed that both the receiver and the sender are in a synchronous message communication phase. This behavior is not acceptable for a low-level communication primitive because we assume that asynchronous and synchronous message passing primitives may be simultaneously issued by a high-level communication library in a multi-threaded environment.

In remote memory write, the receiver's receive address is always explicitly defined by the sender. Message retransmission and flow control due to the lack of receiver's buffer is avoided and thus message transfer overhead is reduced. On the other hand, the sender node must have information about the receiver's pinned-down area before sending a message. Message passing based on remote memory write is performed as follows:

1) A memory area, where a message is to be received, is pinned down and mapping information for the area is sent to the sender by the receiver.

2) The sender receives the mapping information.

3) The sender's memory area is pinned down. The communication handler maintains a page table entry to translate a virtual address to its physical address.

4) To pass the message, the sender passes the information of the sender's data address and receiver's mapping information to the Myrinet communication handler.

5) The Myrinet communication handler sends a message, whose header has the receiver's mapping information, by triggering the DMA function at the sender.

6) When the receiver receives the message, the Myrinet communication handler stores data according to the mapping information.

7) The pinned-down area for receiving a message is released at the receiver.

8) The pinned-down area for sending a message is released at the sender.

In the protocol above, pin-down procedures and negotiations between sender and receiver in steps 1), 2), 3), 7), and 8) are not needed if the same memory areas always participate in message transfers. It seems that procedures 1), 2), and 3) are only need to be done during program initialization. However, we do not employ it because i) one application occupying huge physical memory is not acceptable because we assume that multiple users' applications are running in a time sharing fashion on our cluster environment, and ii) the Myrinet communication handler would have to keep a huge table for the address map in order to allow DMA.

If pin-down and release kernel primitives were issued for each data transfer, higher bandwidth would not be achieved because the cost of pin-down kernel primitive is relatively high. To overcome the problem, the pin-down cache technique is proposed in the next section.

4. Pin-Down Cache

The overhead of the pin-down operation can be reduced if an application program repeatedly transfers data from the same memory area without releasing the pinned-down area. When a request to release a pinned-down memory area is issued, the actual release is postponed until total physical memory reserved for pinned-down areas is exceeded. If the release primitive for pinned-down area has been issued but in fact the area remains pinned down, a subsequent request to pin down the same area can be answered immediately, avoiding the overhead of a pin-down kernel primitive call. We call this technique *Pin-Down Cache*. Pin-down cache works as follows:

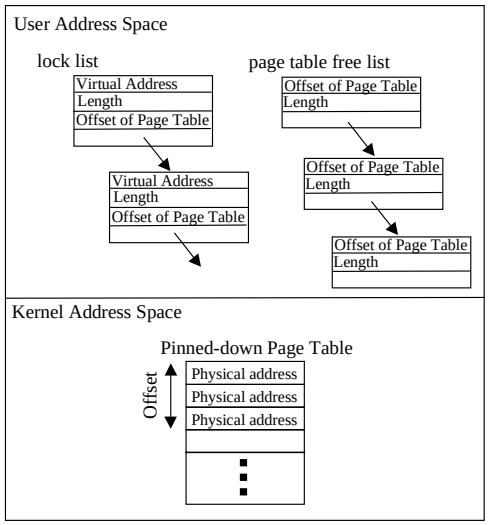

Figure 4. Data Structure

- At a pin-down request,
  1. Check whether or not the requested page has been pinned down.
  2. If the page has been pinned down, the pin-down information is updated and the call returns.
  3. If the page has not been pinned down,
     (a) Check whether or not the total pinned-down area exceeds the maximum size allowed to the user process, and check whether or not the page table kept by the communication handler is full. If either condition is true, some pinned-down area(s) are released based on some replacement algorithm such as LRU.
     (b) The requested area is pinned down.
- At a release request,
  1. Actual pin-down operation is postponed. The entry of the pin-down table for the requested page area is marked for release.

In this way the pin-down cache postpones the actual pin-down release operation. If the maximum pinned-down area available is larger than the aggregate size of the application's data transfer areas, the pin-down cache always hits. Except for the overhead of checking the cache, this becomes exactly as though data transfer area(s) were pinned down statically.

5. Implementation

In PM, the zero copy communication using pin-down cache is realized by a user-level library, a device driver of the operating system kernel, and a Myrinet communication handler. Figure 4 shows the data structures used in the implementation. The physical address information of the pinned-down area is kept by the pinned-down page table in the Myrinet interface memory so that the user cannot directly modify the table. The pinned-down page table is accessed by the Myrinet network handler. In the user memory space, there are i) the *pinned-down area list*, each entry of which contains the virtual address, length and an offset into the pinned-down page table for the pinned-down area, and ii) the free list of the pinned-down page table.

In the current implementation, there are 1,024 page table entries for the Myrinet communication handler and the page size is 4 KBytes. Thus, the user may pin at most 4 MBytes area down at a time.

Device Driver The device driver provides the basic mechanism of pin-down/release operations for communication. The mechanism is realized by the Unix I/O control facility. The VM primitives of NetBSD kernel are used to manage the page table used by the communication handler and pin-down/release pages.

User-Level Library The user-level library realizes the pin-down cache strategy and remote memory write operations. Operations `_pmMLock` and `_pmMUnlock` support pin-down/release primitives with the pin-down cache strategy by using the pinned-down area list and the pin-down page table free list. The pin-down cache replacement algorithm is based on LRU.

Three remote memory write operations, `_pmVWrite`, `_pmWriteDone`, and `_pmSendDone`, are provided. Operation `_pmVWrite` only enqueues the DMA request to the Myrinet interface. That is, when it returns it does not guarantee that the data transfer is finished. Operation `_pmWriteDone` guarantees that previously written user data has been transferred to the memory area in the network interface. The operation does not guarantee that data has been transfered to the remote machine. Operation `_pmSendDone` guarantees that the receiver has received the message and stored the data in the destination.

Myrinet Communication Handler The LANai processor on the Myrinet interface performs the DMA transfer between host memory and Myrinet SRAM area according to the page table information. It also manages sending/seceiving messages to/from other nodes.

6. Evaluation

Pin-down overhead, PM zero-copy message transfer bandwidth, and MPI bandwidth on top of PM are shown. Moreover, results of NAS parallel benchmark suites using MPI are shown. The evaluation was done using our *RWC PC Cluster II* consisting of 64 Pentium Pro 200 MHz processors running NetBSD/i386 1.2.1.

6.1. Basic Performance

Table 2 shows the overhead of the pin-down cache operations. In this evaluation, pin-down/release operations, i.e., `_pmMLock` and `_pmMUnlock`, are issued 100,000 times for the same memory area and the result time is divided by 100,000. In the evaluation of pin-down cache misses, the program is modified so that the pin-down cache always misses.

Table 2. Pin-down Overhead

Pin-down Cache	Time (μseconds)
Miss	$40 + 7.0 \times (number of pages)$
Hit	0.53

The result shows that the pin-down cache miss pays a significant penalty compared with the data transfer cost, i.e., one page length transfer (4 KBytes) on Myrinet only takes 25.6μ seconds while the pin-down cost is more than 47 μ seconds. Of course, the pin-down cost depends on processors and operating systems.

The bandwidth of zero copy message transfer is shown in Figure 5. In this evaluation, transmitted data size varies from 4 bytes to 1 MBytes. The bandwidth for each data size is calculated as the result of 100,000 times transfers. In this evaluation, the receiver pins the receiving memory area once and sends the information to the sender. The pinned-down area is reused during the evaluation. The sender calls the pin-down/release operations each time. By modifying the source program, the pin-down cache hit ratio varies from 0% to 100%.

The maximum bandwidth is 108.8 MBytes/sec for 1 MBytes for a 100 % pin-down cache hit ratio. When pin-down cache is always missed, the maximum bandwidth is still 78.7 MBytes/sec. Compared to old implementations, when a message is larger than 8 KBytes, the bandwidth of zero copy message transfer is higher than the bandwidth of data transfer with data copy, i.e., using the PM message buffer. But it is always lower than the bandwidth of message transfer without data copy. This penalty is due to the overhead of virtual to physical address translation and the

pin-down cache operation, even when the pin-down cache hits 100%.

Note that the one-way message cost for 4 bytes message is 7.5 μ seconds[1]. This performance for such a short message is realized by the PM asynchronous communication facility, i.e., data copy to a message buffer.

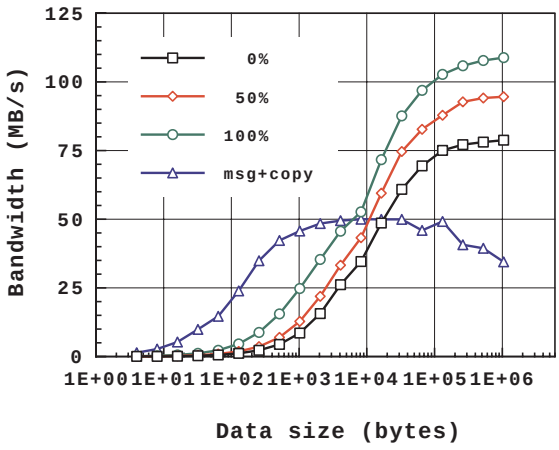

Figure 5. Bandwidth of zero copy memory transfer in PM 1.2

6.2. MPI Performance

MPI has been implemented using the MPICH implementation on top of PM[7, 8]. We implemented two versions of MPI: i) MPI implemented using the PM 1.2 asynchronous communication primitives and ii) MPI implemented using the PM 1.2 zero copy message transfer primitive. Those implementations are called old MPI and new MPI in this paper. According to the MPICH implementation, if the data transfer size is less than or equal to 8 KBytes, asynchronous communication primitives are used in both two versions. Figure 6 shows the bandwidth of synchronous message send/receive of MPI.

In this evaluation, the pin-down cache hit ratio varies between 0 % and 100 %. For comparison, old MPI data transfer bandwidth, `msg+copy` in the figure, is also shown in this figure. The asynchronous message transfer is better performance than the zero copy message transfer for smaller than 8 KBytes message size in case of 100 % cache hit. In case of 0 % cache hit, the zero copy message transfer gives better performance than asynchronous message transfer for message sizes larger than 20 KBytes.

[1]This one-way latency is calculated as half of the round-trip time.

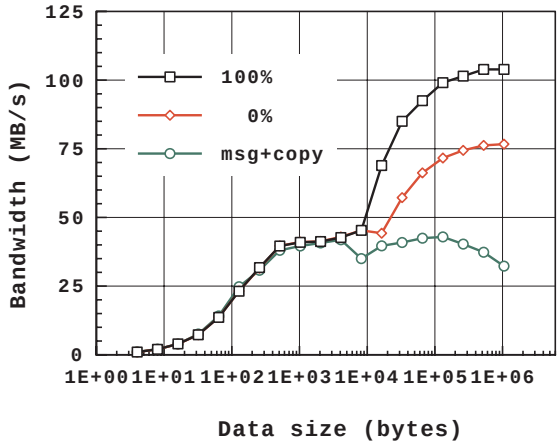

Figure 6. New MPI Bandwidth

Table 3. NAS Parallel Benchmark (Class A) Result

64 PEs

prog.	Message (Mop/s)	VWrite (Mop/s)	speed up	hit	miss	ratio %	size (KB)
IS	69.5	81.9	1.18	802	80	90.9	8
FT	799.7	956.8	1.20	161	847	16.0	32

The maximum bandwidth of MPI using the old PM is about 42.8 MBytes/sec for 128 KBytes while the new MPI achieves 71.6 MBytes/sec(pin-down cache 0% hit) to 99.1 MBytes/sec(100%) for the same data size. Further, the peek bandwidth of new MPI is 103.8 MB/s for 1 MBytes. The MPI bandwidth using zero copy message transfer is lower than the PM zero copy message transfer shown in Figure 5. The reason is that a pin-down operation and a sender/receiver negotiation are performed in a MPI data transfer.

The IS and FT benchmarks from the NAS parallel benchmark[1] NPB2.3 is used to evaluate MPI on PM 1.2. In this evaluation, benchmark programs written in Fortran are automatically translated to C codes. The C codes were compiled by GNU gcc version 2.7.2.1 with -O4 option.

The 64 processors' result is shown in Table 3. Column *Message* is the total performance result (in Mop/s) of the old MPI while column *VWrite* is the time for the new MPI. The speed up column shows how much the new MPI is faster than the old MPI. The pin-down cache statistics are shown in columns *hit*, *miss*, *ratio*. The mean message size is shown in columns *size*.

Although the cache miss ratio is very high in FT, the performance is better than the old MPI implementation. The reason is that the transfer bandwidth for more than 16 KBytes messages is better than the old one as shown in

Figure 6, and the mean message length on FT is 32 KBytes on 64 processors.

The IS and FT benchmark class A programs under the new MPI run from 1.18 to 1.20 times faster since the communication cost is relatively high in those benchmarks on the Pentium Pro 200 MHz.

7. Related Works

According to our design classification of zero copy message transfer techniques described in section 3, VMMC-2[3] and BIP(Basic Interface for Parallelism)[2] of the LHPC are implemeneted based on rendezvous. The VMMC-2 introduces the redirection technique if the receiver has not issued the receive primitive when a message arrives.

To pin-down the user virtual address space, the VMMC-2 is also the same technique in terms of the page table management by kernel primitives and user library. As long as we understand, the pin-down cache technique is not employed in the VMMC-2 but some page table cache technique for the Myrinet SRAM and host memory is employed in the VMMC-2.

The VMMC-2 implementation is based on the same hardware of our PC cluster. The maximum bandwidth is 84 Mbytes/s and a one-way latency is 20 μ seconds in VMMC-2 while the PM achieves 108.8 MBytes/sec for 1 MBytes in case of a 100 % pin-down cache hit ratio and 78.7 MBytes/sec in case of all pin-down cache miss. The 7.5 μ seconds one-way latency is achived in the PM.

The BIP(Basic Interface for Parallelism) is implemented on Pentium Pro (200MHz) with Myrinet and Linux. They achieved 4.3 μ seconds latency (data size is zero) and 126 MBytes/sec bandwidth for 8 Mbytes data size. As long as we know, the detailed design has not been published.

8. Conclusion

We have proposed the technique of zero copy message transfer with *pin-down cache* to eliminate data copy between user specific data area and message buffer. The *pin-down cache* is a technique to reuse the pinned-down area to decrease the number of pin-down and release operations since the pin-down/release kernel primitives are very high cost. The proposed technique has been implemented in PM 1.2.

In PM 1.2, the maximum bandwidth is 109 MBytes/sec for 1 MBytes in case of 100 % pin-down cache hit ratio. In case that pin-down cache is always miss, the maximum bandwidth is still 78.7 MBytes/sec which is higher than the bandwidth of data transfer with data copy, i.e., using the PM message buffer. PM also achieves a low latency, 7.5 μ seconds one-way latency.

The results of the IS and FT benchmark from the NAS parallel benchmark 2.3 have been shown to demonstrate the effectiveness of pin-down cache. Although the cache miss ratio is very high in FT, the performance is better than the old MPI implementation. The IS and FT benchmark class A programs under the new MPI run from 1.18 to 1.20 times faster since the communication cost is relatively high in those benchmarks on the Pentium Pro 200 MHz.

We are currently extending the *RWC PC Cluster II* to 128 Pentium Pro 200 MHz processors and a software DSM to investigate the impact of the zero copy message transfer with the pin-down cache technique.

Readers interested in our clusters may visit the following URL:
`http://www.rwcp.or.jp/lab/pdslab/`
We are currently distributing PM 1.2 and MPC++ MTTL on PM. The other softwares, MPI on PM and SCore-D[5], will soon be distributed.

References

[1] http://science.nas.nasa.gov/Software/NPB/.

[2] http://lhpca.univ-lyon1.fr/bip.html.

[3] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient Support for Reliable, Connection-Oriented Communication. In *HOT Interconnects V*, pages 37–46, 1997.

[4] A. Hori, H. Tezuka, and Y. Ishikawa. Global State Detection using Network Preemption. In *IPPS'97 Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.

[5] A. Hori, H. Tezuka, Y. Ishikawa, N. Soda, H. Konaka, and M. Maeda. Implementation of Gang-Scheduling on Workstation Cluster. In D. G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 76–83. Springer-Verlag, April 1996.

[6] Y. Ishikawa. Multi Thread Template Library – MPC++ Version 2.0 Level 0 Document –. Technical Report TR–96012, RWC, September 1996. *This technial report is obtained via http://www.rwcp.or.jp/lab/mpslab/mpc++/mpc++.html*.

[7] F. B. O'Carroll, A. Hori, Y. Ishikawa, and S. Matsuoka. Implementing MPI in a High-Performance, Multithreaded Language MPC++. In *SWoPP'96*, pages 141–146, 1996.

[8] F. B. O'Carroll, A. Hori, H. Tezuka, Y. Ishikawa, and S. Matsuoka. Performance of MPI on Workstation/PC Clusters using Myrinet. In *Cluster Computing Conference 1997*, 1997.

[9] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking '97*, 1997.