Implementing MPI with the Memory-Based Communication Facilities on the SSS–CORE Operating System[⋆]

Kenji Morimoto, Takashi Matsumoto, and Kei Hiraki

Department of Information Science, Faculty of Science
University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113–0033, Japan
{morimoto, tm, hiraki}@is.s.u-tokyo.ac.jp

Abstract. This paper describes an efficient implementation of MPI on the Memory-Based Communication Facilities; *Memory-Based FIFO* is used for buffering by the library, and *Remote Write* for communication with no buffering. The Memory-Based Communication Facilities are software-based communication mechanisms, with off-the-shelf Ethernet hardware. They provide low-cost and highly-functional primitives for remote memory accesses.
The performance of the library was evaluated on a cluster of workstations connected with a 100Base-TX network. The round-trip time was 71 μs for 0 byte message, and the peak bandwidth was 11.86 Mbyte/s in full-duplex mode. These values show that it is more efficient to realize the message passing libraries with the shared memory model than with the message passing model.

1 Introduction

MPI is a widely used standard library for writing message passing programs, especially on parallel machines with distributed memory [1]. It has been implemented on various platforms such as clusters of workstations and MPPs. In the message passing model, a communication path is established between each pair of tasks, and communication among tasks is performed by applying *send* and *receive* operations to those paths. This model is an abstraction of actual communication paths. In the shared memory model, on the other hand, address spaces of all tasks are mapped into a unified address space, and *load* and *store* operations[1] to that shared space correspond to communication. This model considers an address space as an object, and is called ‘memory-based’.

When these two models are considered as communication models, they are exchangeable, that is, one can emulate the other. Thus the two models are equivalent in expressiveness. So far, the message passing model is widely used because

[⋆] This work is partly supported by Information-technology Promotion Agency (IPA) Japan and by Real World Computing Partnership (RWCP) Japan.

[1] These operations are not necessarily fine-grain memory accesses by a *load* or *store* machine instruction.

that model is believed to be more efficient, and many implementations are supplied as libraries. However, when considered as a functionality provided by a system (hardware and operating system), it has been argued that the shared memory model is superior to the message passing model from the viewpoint of optimization, efficiency, and flexibility [2]. This is because shared memory communication directly utilizes architectural support such as MMU. For this reason, parallel machines should provide an efficient communication functionality based on the shared memory model, rather than the message passing model.

The SSS–CORE is a general-purpose massively-parallel operating system [3, 4] being developed at our laboratory. The Memory-Based Communication Facilities (MBCF) [5, 6] are provided in the SSS–CORE as mechanisms for accesses to remote memory. The MBCF directly support programs which are written based on the shared memory model.

In this paper, communication functions of MPI Ver. 1.1 [1] were implemented with the MBCF on a cluster of workstations. The rest of this paper is organized as follows. Section 2 gives introduction of the MBCF. A detailed explanation of the implementation of MPI is presented in Sect. 3. Section 4 shows performance of the implemented library. We discuss related works in Sect. 5, and conclude in Sect.6.

2 MBCF: Memory-Based Communication Facilities

2.1 Features of MBCF

The MBCF is a software-based mechanism for accessing remote memory. Details of the MBCF are as follows.

1. *Direct access to remote memory*
   The MBCF communicates by reading from and writing to remote memory directly, not by sending messages to some fixed buffer for communication. At the same time, by using architectural memory management mechanisms such as MMU, protection of memory and abstraction of accesses are achieved at clock-level speed.
2. *Use of off-the-shelf network hardware*
   The MBCF is a software-based mechanism, and does not need such dedicated communication hardware as many of MPPs have. In order to achieve high performance, however, it is preferable that the following mechanisms are provided.
   – Switching of address spaces with low overhead
   – TLB which allows many contexts to be mixed
   – Page alias capability
   – High-speed processor cache with physical address tags
   All of these mechanisms are available on most of the latest microprocessors.
3. *Channel-less communication*
   Unlike channel-oriented communication such as user memory mapping with the Myrinet [7], the MBCF achieves abstraction and protection dynamically by system calls for communication.

4. *Highly-functional operations for remote memory*
   Since a receiver of an MBCF packet handles it with software, such compound operations as *swap*, *FIFO write*, and *fetch and add* are available as well as *read* and *write*.
5. *Guarantee for arrival and order of packets*
   Because a system takes care of lost or out-of-order packets, users can make communication as if they were on a reliable network.

2.2 Performance of MBCF

The performance of the MBCF was evaluated on a cluster of workstations connected with a 100Base-TX network. The following machines were used for measurement; Axil 320 model 8.1.1 (Sun SPARCstation 20 compatible, 85 MHz SuperSPARC $\times$ 1), Sun Microsystems Fast Ethernet SBus Adapter 2.0 on each workstation, SMC TigerStack 100 5324TX (non-switching HUB), and Bay Networks BayStack 350T (switching HUB with full-duplex mode). The one-way latency and the peak bandwidth between two nodes were measured.

The one-way latency is the time from the invocation of a system call for remote write to the arrival of the data at the destination task (including the overhead of reading the data). Table 1 shows the one-way latency of *Remote Write* (MBCF_WRITE) and *Memory-Based FIFO* (MBCF_FIFO) for various data-sizes on a HUB.

Table 1. One-way latency of MBCF's remote accesses with 100Base-TX

data-size (byte)	4	16	64	256	1024
MBCF_WRITE (μs)	24.5	27.5	34	60.5	172
MBCF_FIFO (μs)	32	32	40.5	73	210.5

The peak bandwidth is measured by invoking remote accesses continuously. Table 2 shows peak bandwidth of *Remote Write* and *Memory-Based FIFO* for various data-sizes on a HUB (half-duplex) and on a switching HUB (full-duplex).

Both of the results show that the performance of the MBCF is very close to that of the network itself. The MBCF is superior in efficiency to the communication functions of MPPs, which have dedicated communication hardware of higher-potential [6].

3 Implementation of Communication Functions

In this section, the details of the implementation of two point-to-point communication functions, `MPI_Isend()` and `MPI_Irecv()`, are described. All other communication functions can be explained on the analogy of these two functions.

Table 2. Peak bandwidth of MBCF's remote accesses with 100Base-TX

data-size (byte)	4	16	64	256	1024	1408
MBCF_WRITE, half-duplex (Mbyte/s)	0.31	1.15	4.31	8.56	11.13	11.48
MBCF_FIFO, half-duplex (Mbyte/s)	0.31	1.14	4.30	8.53	11.13	11.45
MBCF_WRITE, full-duplex (Mbyte/s)	0.34	1.27	4.82	9.63	11.64	11.93
MBCF_FIFO, full-duplex (Mbyte/s)	0.34	1.26	4.80	9.62	11.64	11.93

The behavior of these two functions varies according to the order of matching two invocations of send and receive functions. In the followings, two cases are described separately; the case where the invocation of `MPI_Isend()` precedes that of `MPI_Irecv()` and the reversed case[2].

3.1 The Case Where `MPI_Isend()` Precedes `MPI_Irecv()`

When `MPI_Isend()` gets started before the invocation of the corresponding `MPI_Irecv()`, the sender does not know which buffer the receiver specifies for incoming data. Therefore the sender should transmit a message to some fixed buffer in the receiver's address space[3]. The MBCF's *Memory-Based FIFO* (*MB_FIFO*) is used for this fixed buffer.

MB_FIFO is one of variations of *Remote Write* (stated in Sect. 3.2). The buffer for a FIFO-queue is taken from the receiving-user's address space, which is specified by the user in advance. The user can make as many queues as space permits. The sender transmits an *MB_FIFO* packet, designating the objective queue by the destination address. The trap handler (managed by the system) on the receiver's side tries to enqueue that packet, and notifies the sender whether the trial succeeded or not, if it is required by the sender to return the state.

In this implementation, the number of *MB_FIFO*'s queues for messages corresponds to the number of processes in the group of `MPI_COMM_WORLD`. When the sender's rank in `MPI_COMM_WORLD` is i, the sender transmits a message to the receiver's i-th queue. The receiver searches the i-th queue for a message from the sender of rank i (in `MPI_COMM_WORLD`). By preparing many queues, it becomes easier to examine message matching and to manage the order of messages.

The sequence of communication is shown in Fig. 1. The left side is the sender's execution flow, and the right is the receiver's. At first the sender enqueues a message to the receiver's *MB_FIFO* queue (because no matching special message has come from the receiver; see Sect. 3.2), and then the receiver dequeues that message when it becomes necessary.

[2] When `MPI_Isend()` and `MPI_Irecv()` are invoked at the same time, it follows the former case. This is detected by assigning sequential numbers to messages.

[3] The sender can choose to transmit a special short message to the receiver (without transmitting the actual message), which tells the receiver to take the message away with *remote read* by itself. This protocol is suitable only for long messages.

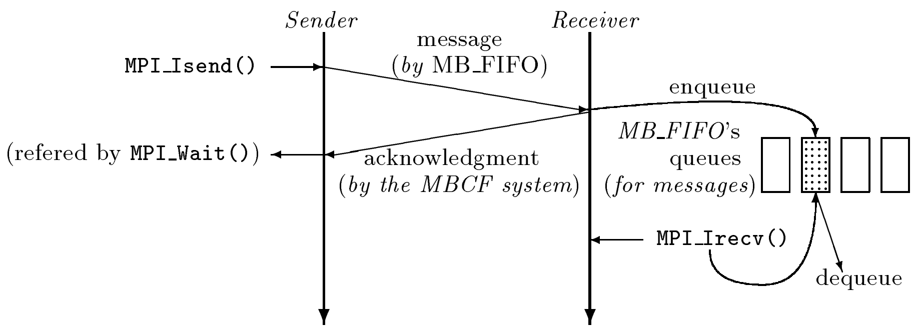

Fig. 1. Communication sequence where `MPI_Isend()` precedes `MPI_Irecv()`

The message may be divided into pieces before transmission owing to the constraint of communication hardware. In order to reduce acknowledgments, notifications of successful enqueueing are not issued, except for the notification for the last packet in a message.

3.2 The Case Where `MPI_Irecv()` Precedes `MPI_Isend()`

When `MPI_Irecv()` is invoked before `MPI_Isend()`, the receiver is able to tell the address of the receiving buffer to the sender, so that the sender can transmit a message directly to that buffer by *Remote Write*. To notify the invocation of `MPI_Irecv()`, another set of *MB_FIFO* queues is used in addition to one used for buffered messages in Sect. 3.1.

The sender of *Remote Write* transmits an MBCF packet with an argument of the destination address represented in receiver's virtual address space. The trap handler on the receiver's side tries to write that packet to the specified address, and returns the resulting state if necessary.

Figure 2 shows the communication sequence. At first the receiver enqueues a special message to the sender's *MB_FIFO* queue which requests the sender to transmit a message directly (because no matching message has come from the sender). `MPI_Isend()` on the sender checks queues of requests for sending, and responds to a matching request. For this *Remote Write*, acknowledging messages are not needed because it is guaranteed that the *Remote Write* succeeds.

4 Performance

This section shows the performance of the MPI library implemented on the SSS–CORE with the MBCF. The performance was evaluated on a cluster of workstations connected with a 100Base-TX network. The same equipments were used as in Sect. 2.2.

For comparison, the performance of the MPICH Ver. 1.1 [8] on the SunOS 4.1.4 was also evaluated with the same system configuration. The MPICH is an

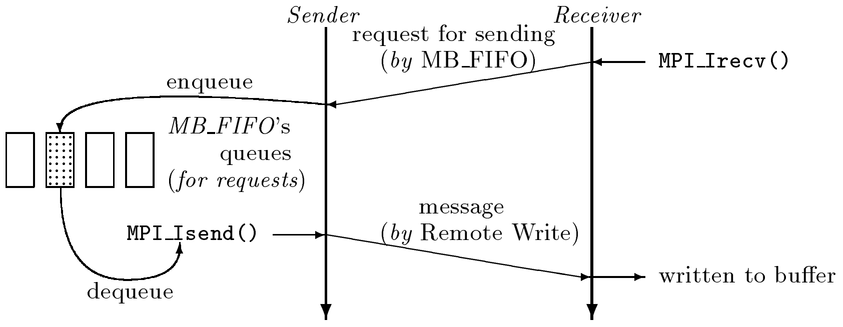

Fig. 2. Communication sequence where `MPI_Irecv()` precedes `MPI_Isend()`

implementation of MPI being developed at the Argonne National Laboratory and the Mississippi State University. The MPICH on a cluster of workstations uses TCP sockets for communication, which are based on the message passing model. For the SunOS 4.1.4, the network switch can not be used in full-duplex mode owing to the constraint of the device driver.

Let our implementation be called “MPI/MBCF,” and the MPICH be called “MPICH/TCP.” In order to see the effect of using *Remote Write*, there are two different versions for MPI/MBCF implementation. One issues “requests for sending (SendReqs)” as stated above, and another does not. The latter treats `MPI_Isend()` and `MPI_Irecv()` as if `MPI_Isend()` always preceded `MPI_Irecv()`. As a consequence, there are five cases, as shown below.

NSRH	MPI/MBCF without SendReqs in half-duplex mode (HUB)
SRH	MPI/MBCF with SendReqs in half-duplex mode (HUB)
NSRF	MPI/MBCF without SendReqs in full-duplex mode (switching HUB)
SRF	MPI/MBCF with SendReqs in full-duplex mode (switching HUB)
MPICH	MPICH/TCP in half-duplex mode (HUB)

First we measured the round-trip time. In the evaluation program, two processes issue `MPI_Irecv()`'s in advance. And then one process calls `MPI_Isend()`, `MPI_Wait()` (for sending), and `MPI_Wait()` (for receiving), and another does `MPI_Wait()` (for receiving), `MPI_Isend()`, and `MPI_Wait()` (for sending). The time in the former process from the beginning of `MPI_Isend()` to the end of `MPI_Wait()` (for receiving) is measured as a round-trip time. For MPI/MBCF, the time is measured for every iteration by $0.5\,\mu$s-resolution counter, and the minimum value is taken as a round-trip time. For MPICH/TCP, unfortunately, this counter is not available, and the minimum value of average times is taken.

Table 3 shows the round-trip time for various message sizes. Since there is large penalties and little benefit to the round-trip time in full-duplex mode, NSRF and SRF are omitted. NSRH and SRH are much faster than MPICH. The difference between NSRH and SRH is caused by the difference between *MB_FIFO* and *Remote Write*; *Remote Write* is faster, and much easier to manage because it does not require explicit acknowledgments. Because the MPI library needs additional operations such as message matching, the SRH's round-trip time 71 μs is within a reasonable range compared with the MBCF's one-way latency 24.5 μs.

Table 3. Round-trip time of MPI with 100Base-TX

message size (byte)	0	4	16	64	256	1024	4096
NSRH (μs)	112	137	139	154	223	517	1109
SRH (μs)	71	85	85	106	168	438	1026
MPICH (μs)	968	962	980	1020	1080	1255	2195

The peak bandwidth was measured for various message sizes by sending messages in one direction, shown in Table 4. MPI/MBCFs gained higher bandwidth than MPICH/TCP, especially for small messages. The difference between NSRH and SRH reveals that the additional packets of "requests for sending" interfere with the communication of actual data. In full-duplex mode, however, the undesirable influence of "requests for sending" is not so clear as in half-duplex mode.

Table 4. Peak bandwidth of MPI with 100Base-TX

message size (byte)	4	16	64	256	1024	4096	16384	65536	262144	1048576
NSRH (Mbyte/s)	0.14	0.54	1.89	4.92	8.54	10.21	10.34	10.43	10.02	9.96
SRH (Mbyte/s)	0.14	0.53	1.82	4.72	8.08	9.72	10.15	9.78	9.96	10.00
NSRF (Mbyte/s)	0.15	0.59	1.98	5.51	10.58	11.70	11.78	11.81	11.82	11.82
SRF (Mbyte/s)	0.14	0.57	1.90	5.33	10.22	11.68	11.77	11.85	11.85	11.86
MPICH (Mbyte/s)	0.02	0.09	0.35	1.27	3.54	6.04	5.59	7.00	7.77	7.07

5 Related Work

To reduce the buffering overhead, MPIAP [9] uses *put* and *get* of AP3000, and CRI/EPCC MPI [10] uses Shared Memory Access of T3D. They send a special message from the sender, perform message matching only on the receiver's side, and issue remote read. This method increases latency. MPI-EMX [11] uses *remote memory write* of EM-X, but all of them are implementations with dedicated communication hardware on MPPs.

6 Summary

An MPI library has been implemented for the SSS–CORE operating system by the use of the MBCF. The MBCF's *MB_FIFO* is used for buffering by the library, and *Remote Write* for direct transmission with no buffering. When the receiver precedes, *Remote Write* reduces the latency of communication. Even when the sender precedes, *MB_FIFO* efficiently transmits the message to the receiver.

The performance of the library was evaluated on a cluster of workstations connected with a 100Base-TX network. The round-trip time was 71 μs (for 0 byte message) and the peak bandwidth was 10.15 Mbyte/s (in half-duplex mode) and 11.86 Mbyte/s (in full-duplex mode). These values show that the additional overhead of the library is small, and that it is efficient to use the MBCF as a base of mechanisms for message passing. More generally, it is effective to use software-implemented virtual-address-based shared memory communication as a base of a communication system of parallel machines with distributed memory.

References

[1] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. `http://www.mpi-forum.org/`, June 1995.

[2] T. Matsumoto and K. Hiraki. Shared memory vs. message passing (in Japanese). In *IPSJ SIG Reports 97-ARC-126, Vol. 97, No. 102*, pages 85–90, October 1997.

[3] T. Matsumoto, S. Furuso, and K. Hiraki. Resource management methods of the general-purpose massively-parallel operating system: SSS–CORE (in Japanese). In *Proc. of 11th Conf. of JSSST*, pages 13–16, October 1994.

[4] T. Matsumoto, S. Uzuhara, and K. Hiraki. A general-purpose scalable operating system: SSS–CORE. In *Proc. of 20th Int. Conf. on Software Engineering (2)*, pages 147–152, April 1998.

[5] T. Matsumoto, T. Komaarashi, S. Uzuhara, S. Takeoka, and K. Hiraki. A general-purpose massively-parallel operating system: SSS–CORE — implementation methods for network of workstations — (in Japanese). In *IPSJ SIG Notes 96-OS-73, Vol. 96, No. 79*, pages 115–120, August 1996.

[6] T. Matsumoto and K. Hiraki. MBCF: A protected and virtualized high-speed user-level memory-based communication facility. In *Proc. of Int. Conf. Supercomputing '98 (to be appeared)*, July 1998.

[7] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An operating system coordinated high performance communication library. In B. Hertzberger and P. Sloot, editors, *High-Performance Computing and Networking*, volume 1225 of *Lecture Notes in Computer Science*, pages 708–717. Springer Verlag, April 1997.

[8] 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, September 1996.

[9] D. Sitsky and P. Mackerras. System developments on the Fujitsu AP3000. In P. Mackerras, editor, *Proc. of 7th Parallel Computing Workshop*, September 1997.

[10] K. Cameron, L. Clarke, and G. Smith. CRI/EPCC MPI for CRAY T3D. `http://www.epcc.ed.ac.uk/t3dmpi/Product/`, September 1995.

[11] O. Tatebe, Y. Kodama, S. Sekiguchi, and Y. Yamaguchi. Efficient implementation of MPI using remote memory write (in Japanese). In *Proc. of Joint Symp. on Parallel Processing '98*, pages 199–206, June 1998.