Performance Evaluation of MPI/MBCF with the NAS Parallel Benchmarks

Kenji Morimoto, Takashi Matsumoto, and Kei Hiraki

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

Abstract. MPI/MBCF is a high-performance MPI library targeting a cluster of workstations connected by a commodity network. It is implemented with the Memory-Based Communication Facilities (MBCF), which provides software mechanisms for users to access remote task's memory space with off-the-shelf network hardware. MPI/MBCF uses *Memory-Based FIFO* for message buffering and *Remote Write* for communication without buffering from among the functions of MBCF. In this paper, we evaluate the performance of MPI/MBCF on a cluster of workstations with the NAS Parallel Benchmarks. We verify whether a message passing library implemented on the shared memory model achieves higher performance than that on the message passing model.

1 Introduction

The message passing model is a programming paradigm for a parallel processing environment. The Message Passing Interface (MPI) [3] is an interface-level instance of it. The shared memory model is another programming paradigm. When these two models are considered as communication models, they are exchangeable; one can emulate the other. In regard to implementation, a shared memory communication mechanism has advantages in performance over a message passing communication mechanism. This is because the former can transfer data directly to the destination memory while the latter needs intermediate buffering space. With the help of memory-oriented architectural supports such as MMU, the mechanism of shared memory communication can be implemented with lower software overheads than that of message passing communication even on packet-based off-the-shelf network hardware.

From the above consideration, we hold that message passing communication implemented with a high-performance shared memory communication mechanism gives better performance than that implemented with a message passing communication mechanism. We verify our claim by instrumentation of practical parallel applications in this paper. We chose MPI Ver. 1.2 [3, 4] for a message passing communication library to be implemented. We used the Memory-Based Communication Facilities (MBCF) [7, 8] as a software-implemented shared memory communication mechanism. The round-trip time and the peak bandwidth of our MPI library, called MPI/MBCF, show that the MBCF functions for shared memory communication work effectively for the message passing library [10]. We employed the NAS Parallel Benchmarks [1, 2] as benchmark applications to evaluate the behavior of MPI/MBCF in various communication patterns.

The rest of this paper is organized as follows. Section 2 gives the explanation of MBCF and MPI/MBCF, followed by the basic performance parameters of MPI/MBCF. We show the results of performance evaluation of MPI/MBCF with the NAS Parallel Benchmarks in Sect. 3 We conclude in Sect. 4 with a summary.

2 MPI/MBCF

2.1 Implementation of MPI/MBCF

MBCF is a software-based mechanism for accessing a remote memory space even on packet-based off-the-shelf network hardware such as Ethernet. We used *Remote Write* and *Memory-Based FIFO* for our MPI implementation from among the functions of MBCF. *Remote Write* enables a user to write data directly to remote task's logical address space. With *Memory-Based FIFO*, a user can send data to a remote FIFO-queue. The buffering area of the queue is reserved in the remote user's address space.

MPI/MBCF [10] is a complete implementation of MPI-1.2. To implement two fundamental point-to-point communication functions, send and receive, we employed two protocols for actual communication by the library. One is the *eager* protocol in terms of MPICH [5]. The other is the *write* protocol in our terms [9]. These two protocols are described with the behavior of the sender and the receiver as follows.

Eager protocol The sender sends the message header and data to the receiver with a pre-fixed address. The receiver examines matching of the message and pending receives to take out the data.

Write protocol The receiver sends the header to the source process. This header contains the address of the receive buffer. The sender examines matching of the header and pending sends to send the data by a remote write primitive.

The eager protocol enables the sender to send data as soon as the data gets ready. The write protocol enables the receiver to receive data without buffering. We combined these two protocols as follows.

– The receiver sends the header to the source process as a *request for sending* (based on the write protocol), if a matching message has not arrived yet and if the source process is uniquely specified.
– The sender sends the message data to the receiver by a remote write primitive (based on the write protocol) if a matching request has arrived. Or else the sender sends the message header and data to the receiver with a pre-fixed address (based on the eager protocol).

2.2 Basic Performance Parameters of MPI/MBCF

We measured the round-trip time and the peak bandwidth of MPI/MBCF on a cluster of workstations connected with a 100BASE-TX Ethernet. The following machines were used for measurement: Sun Microsystems SPARCstation 20 (85 MHz SuperSPARC $\times$ 1), Sun Microsystems Fast Ethernet SBus Adapter 2.0 on each workstation, and SMC TigerStack 100 5324TX (non-switching Hub). The operating system used for measurement is SSS–CORE Ver. 1.1a [6].

The performance of MPICH 1.1 on SunOS 4.1.4 was also evaluated with the same equipments to make a comparison. MPICH employs TCP sockets for communication when it is used on a cluster of workstations.

In order to examine the effects of the write protocol, two different versions of MPI/MBCF are used for the performance evaluation. One issues requests for sending as explained in Sect. 2.1, and the other does not. The former implementation is denoted by SR and the latter NSR for short in the following. In NSR, the sender always transmits a message with *Memory-Based FIFO* and never with *Remote Write*, like the eager protocol.

From the performance parameters presented in Table 1, it is shown that the latency of MPI/MBCF is ten times smaller than that of MPICH/TCP. MPI/MBCF gains high bandwidth for small messages as well as for large messages. The latency of SR is smaller than that of NSR. In SR, however, request packets from the receiver interfere with message packets from the sender. Thus the bandwidth of SR is smaller than that of NSR.

Table 1. Basic performance parameters of MPI libraries with 100BASE-TX

MPI library	MPI/MBCF		MPICH/TCP
	SR	NSR	
round-trip time for 0 byte message (μs)	71	112	968
peak bandwidth for 256 bytes message (Mbytes/s)	4.72	4.92	1.27
peak bandwidth for 16 Kbytes message (Mbytes/s)	10.15	10.34	5.59

3 Performance Evaluation with the NAS Parallel Benchmarks

3.1 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) [1, 2] is a suite of benchmarks for parallel machines. It consists of the following five kernel programs and three computational fluid dynamics (CFD) applications.

EP Random number generation by the multiplication congruence method

Table 2. Characteristics of NPB Programs with 16 processes

program	EP	MG	CG	IS	LU	SP	BT
communication freq (Mbytes/s)	0.00	20.21	36.24	22.10	5.58	19.50	14.68
communication freq (# of messages/s)	16	13653	6103	2794	5672	1657	2020
Remote Write availability rate (%)	50.51	0.02	47.89	97.15	8.66	42.55	45.22
performance ratio, MPI/MBCF vs. MPICH	1.01	1.42	1.30	1.46	1.36	1.59	1.19
performance ratio, SR vs. NSR	1.00	1.00	1.02	1.44	1.01	1.00	1.01

MG Simplified multigrid kernel for solving a 3D Poisson PDE
CG Conjugate gradient method for finding the smallest eigenvalue of a large-scale sparse symmetric positive definite matrix
FT Fast-Fourier transformation for solving a 3D PDE
IS Large-scale integer sort
LU CFD application using the symmetric SOR iteration
SP CFD application using the scalar ADI iteration
BT CFD application using the 5×5 block size ADI iteration

The NPB 2.x provides source codes written with MPI. Each of the eight problems is classified into five (S, W, A, B, and C) classes according to its problem size configuration.

3.2 Conditions

The same workstations as stated in Sect. 2.2 and 3Com Super Stack II Switch 3900 (switching Hub) were used. SR and NSR of MPI/MBCF on SSS–CORE Ver. 1.1a and MPICH 1.1.1 on SunOS 4.1.4 are compared.

We used gcc-2.7.2.3 and g77-0.5.21 for compilation. Since the FT kernel program cannot be compiled with g77, it is omitted in the following evaluation.

The problem size is fixed to Class W because the programs of Class A cannot be executed on SunOS for comparison owing to the shortage of memory.

3.3 Experimental Results

Table 2 shows the characteristics of the benchmark programs. These were measured on SR with 16 processes by inserting counter codes into MPI/MBCF. The communication frequency is computed from the total amount of data (or the total number of messages) transmitted among all processes divided by the execution time. The *Remote Write* availability rate is computed from the amount of data transmitted with *Remote Write*, divided by the total amount of data. The performance ratio is the reciprocal of the execution time ratio.

Table 3 shows the execution time of EP. 2^{26} random numbers are computed. In EP, interprocess communication occurs only for gathering final results. The amount of transmitted data is very small. Consequently the results of MPI/MBCF and MPICH differ by 1 % or less.

Table 3. Execution time of NPB EP in seconds

# of processes	1	2	4	8	16
SR [speed-up]	120.94 [1.00]	60.50 [2.00]	30.26 [4.00]	15.14 [7.99]	7.57 [15.98]
NSR [speed-up]	120.95 [1.00]	60.51 [2.00]	30.26 [4.00]	15.14 [7.99]	7.57 [15.98]
MPICH [speed-up]	121.17 [1.00]	60.50 [2.00]	30.47 [3.98]	15.22 [7.96]	7.62 [15.90]

Table 4. Execution time of NPB MG in seconds

# of processes	1	2	4	8	16
SR [speed-up]	39.30 [1.00]	23.08 [1.70]	13.52 [2.91]	7.31 [5.38]	4.80 [8.19]
NSR [speed-up]	39.26 [1.00]	23.18 [1.69]	13.48 [2.91]	7.29 [5.39]	4.80 [8.18]
MPICH [speed-up]	39.17 [1.00]	23.61 [1.66]	16.41 [2.39]	9.47 [4.14]	6.81 [5.75]

Table 4 shows the execution time of MG. The problem size is $64 \times 64 \times 64$ and the number of iterations is 40. Point-to-point communication operations for messages of around 1 Kbytes are performed very frequently to exchange data across partitioning boundaries. Since MPI/MBCF is suitable for fine-grain communication as shown in Sect. 2.2, the performance with MPI/MBCF is better by 42 % than that with MPICH. All of the receive functions in MG specify an unnecessary wild card `MPI_ANY_SOURCE` as a source. In SR, this disables the receiver from issuing requests for sending to a unique source. This causes extremely low availability (0.02 %) of *Remote Write*. Thus the results of SR and NSR differ very little.

Table 5 shows the execution time of CG. The problem size is 7000 and the number of iterations is 15. Collective reduction operations and point-to-point communication operations are performed for messages of around 10 Kbytes. Although the communication frequency of CG is high (i.e. hard for MPICH), the message size is large (i.e. easy even for MPICH). Therefore the difference between MPI/MBCF and MPICH in CG is smaller than in MG. *Remote Write* is applied to 48 % of messages in SR so that the performance of SR is improved by 1.7 % from NSR.

Table 6 shows the execution time of IS. The problem size is 2^{20} and the number of iterations is 10. About 1 Mbytes messages are exchanged at each iteration by collective all-to-all communication functions. Because the amount of computation is small, the performance of collective operations dominates the whole performance more and more as the number of processes increases. In MPI/MBCF, functions for collective operations are written so that receive functions are first invoked. Thus, in SR, 97 % of messages are transmitted with *Remote Write*. The performance of SR is improved by 44 % from NSR and by 46 % from MPICH.

Table 7 shows the execution time of LU. The problem size is $33 \times 33 \times 33$ and the number of iterations is 300. Point-to-point communication operations are performed for messages of some hundred bytes. Since the message size is small, MPI/MBCF achieves better performance by 36 % than MPICH. As well as in MG, the use of `MPI_ANY_SOURCE` reduces the use of *Remote Write* to 8.7 % for SR.

Table 8 shows the execution time of SP. The problem size is $36 \times 36 \times 36$ and the number of iterations is 400. Point-to-point communication operations are performed for messages of around 10 Kbytes. MPI/MBCF achieves better performance by 59 % than MPICH. There is little difference between results of SR and NSR, though SR utilizes *Remote Write* for 43 % of messages. This is because communication in SP is organized to hide latency in some degree.

Table 9 shows the execution time of BT. The problem size is $24 \times 24 \times 24$ and the number of iterations is 200. Point-to-point communication operations are performed for messages of around 10 Kbytes. MPI/MBCF achieves better performance by 19 % than MPICH. As well as in SP, there is little difference between results of SR and NSR though SR utilizes *Remote Write* for 45 % of messages.

3.4 Summary of Results

Except for in EP, MPI/MBCF achieves better performance by 19 %–59 % than MPICH/TCP. Especially when small messages are transmitted frequently, the large overheads of MPICH/TCP lower the performance. Thus the difference in performance between two libraries expands, as shown in MG and LU.

The NSR implementation of MPI/MBCF uses *Memory-Based FIFO* alone. It has a resemblance to MPICH/TCP in that both of them are message-based implementations of MPI. The lower latency and higher bandwidth of NSR, which are mainly brought by MBCF, cause better performance by 2 %–60 % than MPICH/TCP on the very same machines.

Compared with NSR, SR utilizes *Remote Write* to communicate without buffering when receive functions are invoked before send functions (e.g. in CG, IS, and LU). Therefore SR achieves better performance in such cases. This shows that the combination of the eager protocol and the write protocol improves the practical performance, as well as the basic performance. When a wild card is specified as a source in the receiver, however, SR cannot use *Remote Write*. This deprives the opportunities of efficient communication in SR.

4 Summary

We have implemented a full MPI-1.2 library, MPI/MBCF, with a shared memory communication mechanism, MBCF. The performance of MPI/MBCF was evaluated on a cluster of workstations connected with a 100BASE-TX Ethernet. By executing the NAS Parallel Benchmarks, it was shown that MPI/MBCF with *Remote Write* achieves better performance than MPI/MBCF without *Remote Write* and than MPICH/TCP. These results give corroborative evidence to our claim; a message passing library achieves higher performance by using a shared

Table 5. Execution time of NPB CG in seconds

# of processes	1	2	4	8	16
SR [speed-up]	69.59 [1.00]	36.20 [1.92]	21.09 [3.30]	11.03 [6.31]	7.74 [8.99]
NSR [speed-up]	69.59 [1.00]	36.84 [1.89]	21.44 [3.25]	11.38 [6.12]	7.87 [8.84]
MPICH [speed-up]	68.05 [1.00]	38.70 [1.76]	23.90 [2.85]	12.74 [5.34]	10.06 [6.76]

Table 6. Execution time of NPB IS in seconds

# of processes	1	2	4	8	16
SR [speed-up]	10.12 [1.00]	7.17 [1.41]	4.49 [2.25]	3.35 [3.02]	1.97 [5.14]
NSR [speed-up]	10.08 [1.00]	7.15 [1.41]	4.82 [2.09]	3.77 [2.67]	2.83 [3.56]
MPICH [speed-up]	8.89 [1.00]	7.10 [1.25]	5.41 [1.64]	4.73 [1.88]	2.88 [3.09]

Table 7. Execution time of NPB LU in seconds

# of processes	1	2	4	8	16
SR [speed-up]	1038.72 [1.00]	535.36 [1.94]	277.25 [3.75]	149.58 [6.94]	80.89 [12.84]
NSR [speed-up]	1028.93 [1.00]	538.76 [1.91]	281.93 [3.65]	151.02 [6.81]	82.08 [12.54]
MPICH [speed-up]	1020.13 [1.00]	558.06 [1.83]	297.38 [3.43]	173.66 [5.87]	110.07 [9.27]

Table 8. Execution time of NPB SP in seconds

# of processes	1	4	9	16
SR [speed-up]	1400.17 [1.00]	343.50 [4.08]	155.65 [9.00]	91.89 [15.24]
NSR [speed-up]	1398.89 [1.00]	344.11 [4.07]	158.39 [8.83]	91.74 [15.25]
MPICH [speed-up]	1392.28 [1.00]	407.87 [3.41]	201.55 [6.91]	146.46 [9.51]

Table 9. Execution time of NPB BT in seconds

# of processes	1	4	9	16
SR [speed-up]	618.00 [1.00]	155.54 [3.97]	71.43 [8.65]	37.66 [16.41]
NSR [speed-up]	618.71 [1.00]	155.82 [3.97]	67.78 [9.13]	37.87 [16.34]
MPICH [speed-up]	616.76 [1.00]	197.20 [3.13]	91.39 [6.75]	44.89 [13.74]

memory communication mechanism than with a message passing communication mechanism.

The experiments were made for two different combinations of operating systems and communication mechanisms on the very same machines with off-the-shelf hardware. These results show that it is possible to achieve large improvement of performance by improving the operating system and the communication library, without modifying applications. This also suggests the effectiveness of a cluster of workstations without dedicated communication hardware.

References

1. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, March 1994.
2. D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, December 1995.
3. Message Passing Interface Forum. MPI: A message-passing interface standard. `http://www.mpi-forum.org/`, June 1995.
4. Message Passing Interface Forum. MPI-2: Extensions to the message-passing interface. `http://www.mpi-forum.org/`, July 1997.
5. 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.
6. 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.
7. T. Matsumoto and K. Hiraki. Memory-based communication facilities of the general-purpose massively-parallel operating system: SSS–CORE (in Japanese). In *Proc. of 53rd Annual Convention of IPSJ (1)*, pages 37–38, September 1996.
8. T. Matsumoto and K. Hiraki. MBCF: A protected and virtualized high-speed user-level memory-based communication facility. In *Proc. of Int. Conf. on Supercomputing '98*, pages 259–266, July 1998.
9. K. Morimoto. Implementing message passing communication with a shared memory communication mechanism. Master's thesis, Graduate School of University of Tokyo, March 1999.
10. K. Morimoto, T. Matsumoto, and K. Hiraki. Implementing MPI with the memory-based communication facilities on the SSS–CORE operating system. In V. Alexandrov and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1497 of *Lecture Notes in Computer Science*, pages 223–230. Springer-Verlag, September 1998.