A Case Study of MPI: Portable and Efficient Libraries*

Christian Bischof† Steven Huss-Lederman‡ Xiaobai Sun†
Anna Tsao‡ Thomas Turnbull‡

Abstract

In this paper, we discuss the performance achieved by several implementations of the recently defined Message Passing Interface (MPI) standard. In particular, performance results for different implementations of the broadcast operation are analyzed and compared on the Delta, Paragon, SP1 and CM5.

1 Introduction

For the past several years, members of the Parallel Research on Invariant Subspace Methods (PRISM) project have been investigating scalable parallel eigensolvers for distributed memory systems [1, 3]. The ultimate objective of this research is the development of portable and efficient libraries for this fundamental numerical linear algebra kernel.

In the course of our work, we, like many other library developers, have been faced with many issues relating to portable programming. Previously, a notable obstacle to library development was the lack of standardization in message passing, from both a programming and a functional point of view. This lack of standardization made it difficult to write portable high performance message passing codes. This difficulty was overcome in a variety of ways. One technique used is to isolate the message passing calls into a limited number of communication routines. The library writer then ports this limited number of routines to machines of interest. Another method is to use the communication routines provided by a third party. Calls to these routines are then either translated into native calls on each machine or into a message passing system provided by the third party. These strategies have been used in many packages including Chameleon [8], Express [14], p4 [4], PARMACS [9] and PVM [2]. A third alternative is to take a code written for one native message passing system and use a look-alike library to allow usage of the same code on another native message passing system (cf. [8, 12]). These solutions have several drawbacks. First, the library writers must either port the communications software themselves or depend on software from a third party. Second, debugging is often complicated by the fact that calls are converted into native calls on each system. Third, the end user often needs to install the communications software on a local machine. Since different libraries may use different

*This work was partially supported by the Applied and Computational Mathematics Program, Advanced Research Projects Agency, under Contract DM28E04120, and by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38. Access to the Intel Touchstone Delta and Paragon systems operated by Caltech on behalf of the Concurrent Supercomputing Consortium was provided by NSF. The authors gratefully acknowledge use of the Argonne High-Performance Computing Research Facility. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

‡Supercomputing Research Center, Bowie, MD 20715-4300.

packages, the user often has to install several packages.

A library writer would prefer to use a standard software library for message passing supported by each vendor in much the same way that codes are written in Fortran or C. The recently completed Message Passing Interface (MPI) standard defines a common set of message passing routines that will be supported by most vendors [13]. Its design allows for implementation on a wide range of platforms, including networks of workstations and homogeneous or heterogeneous Massively Parallel Processors (MPPs). Not only does this standard allow software portability, but it should lead to efficient codes since it will be implemented by each vendor for its own hardware. These considerations alone make MPI a valuable standard. In addition, MPI combines into a single package many useful and powerful features from different systems, as well as including a number of features not currently available in any other package.

Since the final release of the MPI standard in April 1994, a number of MPI implementations have been introduced. The most widely available is the Argonne National Laboratory/Mississippi State (ANL/MS) portable, public domain version [7]. This MPI implementation runs on networks of workstations, the IBM SP1, the Intel Delta and Paragon, the Meiko CS-2, the Thinking Machine CM5 and other machines. In addition, IBM has produced its own version for the SP1 [5, 6]. Cray Research, Intel and Meiko have announced their intention to support MPI on their latest platforms. Other implementations of MPI have also been produced, such as those by Ohio State University and University of Edinburgh.

We have written numerous codes designed to evaluate the performance of different implementations of various collective communication operations. In this paper, we present results for performing a 1-dimensional broadcast across a set of nodes. We chose this collective operation since it is important in a number of applications. Additionally, it is easy to implement but requires specific optimizations on most machines to achieve maximal performance.

In this paper we will discuss the efficiency of several aspects of MPI as implemented by ANL/MS. The efficiency of the ANL/MS MPI implementation will be evaluated by comparing the MPI broadcast collective operation to optimized versions written by us for the Intel Delta using both MPI and the native NX message passing systems. All three implementations of the broadcast operation will be compared on the Delta and Paragon; in addition, the two MPI-based implementations will also be compared on the SP1 and CM5.

2 Description of Tests of Broadcast Implementations

Within MPI, broadcasts can be performed through calls to MPI_Bcast. We have also implemented our own algorithm that performs a broadcast by recursively dividing the list of processors in half and sending to the nearest processor on the other half of the list. When performed over p processors, this algorithm takes $\lceil \log_2 p \rceil$ stages. If the processors are connected in a linear array in the order given in the list passed to the routine, then no contention for wires will occur during the algorithm. A more detailed description of this routine can be found in [10].

For comparison purposes, two or three versions of broadcast were run. The first version simply uses the MPI collective call. The second version implements our algorithm utilizing MPI send and receive calls to perform message passing. The message passing strategy used was highly optimized for the Intel Delta and performs well on the Paragon. The third version, available on the Intel machines, implements our algorithm using Intel's NX message passing system to achieve maximum performance. The only difference between the second and third versions is the message passing calls used. The algorithms we implemented perform best when the processors behave as if they are linearly connected. On the Intel Delta and Paragon this can be achieved by using partitions that are linear in the 2D mesh. Such partitions were used in all tests involving these two machines in order to maximize performance. Our test codes verify that each collective operation performed is reliable and correct. All of our routines are publicly available by sending mail to `prism@super.org`.

On a number of systems, the use of the MPI ready mode (forced-type in Intel NX) yields performance gains. In ready mode, the application must guarantee that a receive is posted before the send occurs. Guaranteeing that the receive is posted is typically done by having the receiver send a zero length synchronization message to the sender once the receive has been posted. Using ready mode allows some systems to avoid using the normal flow control protocols and bypass the use of system buffers. The flow control protocol on the Delta causes forced-type to be approximately 25% faster than regular-type for long messages [15, 11]. The flow control protocol on the Paragon has been improved so that this is no longer the case. However, forced-type can still be a win for longer messages since it avoids system buffering on the receiving end. The use of system buffering necessitates a copy of the transmitted data from the system buffer into the application buffer, which can degrade performance. Counterbalancing this performance increase is the implied synchronization between the send/receive pair in ready mode. Given these considerations, the MPI and NX versions of our routines were tested using both ready mode/forced-type and regular mode message passing calls.

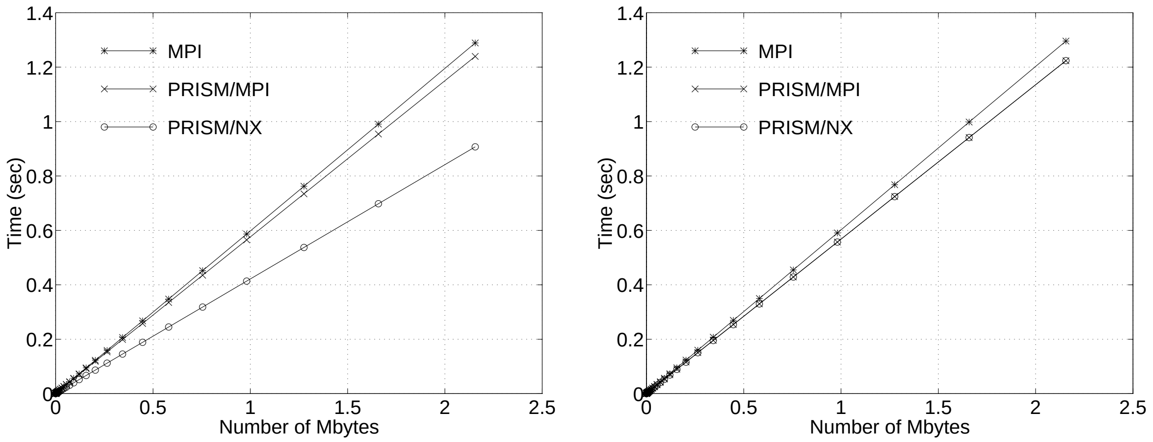
The results given in the next section were obtained on the machines running the operating systems listed in Table 1. Each machine was running other jobs during the testing. To minimize the impact from other users, each case was run three times and the results averaged. All the processors were synchronized before each trial began. Furthermore, when comparing different versions of the same operation, the tests were run at the same time on the same processors. Rerunning the same tests at different times has shown the results to be reproducible. We used the MPI version from ANL/MS dated July 22, 1994, except on the CM5, which required the use of the September 5, 1994 release.

Machine	OS
Delta	NX Release 1.5
Paragon	OSF/1 Rel. 1.0.4 Server 1.2 R1.2
SP1	AIX Version 3.2/eui.1.06.4
CM5 with Viking nodes	CMOST version 7.3, patch 3, CMMD version 3.2 Beta

TABLE 1. *Machines and OS used in testing*

3 Results

Results will first be given for the Intel Delta. This is the machine for which our collective operations were optimized, and thus the performance is expected to be best on this machine. Figure 1 shows the time for performing a broadcast as a function of the amount of data sent across 16 nodes, given in Mbytes. The graph shows the time for the slowest processor. It is seen in Figure 1a that the MPI broadcast (labeled MPI) is only slightly slower than our routine using MPI ready mode for message passing (labeled PRISM/MPI). However, our routine using NX forced-type message passing (labeled PRISM/NX) is about 29% faster for longer messages. Figure 1b shows the same comparisons when ready mode/forced-type is not used by our routines. Notice that the time for the PRISM/MPI routine is unchanged, but the time of the PRISM/NX routine is greater and now identical to the PRISM/MPI routine. This shows that the current ANL/MS MPI gives the same performance as NX for regular sends but not for ready sends. Figure 2 makes the same comparisons as Figure 1a, except the broadcast is now performed over 15 instead of 16 processors. As expected, the PRISM routines take the same amount of time as in the case of 16 nodes. However, the MPI routine is almost twice as slow, although one would expect the same or better performance. This result is seen whenever the number of processors is not a power of two.



a) ready mode *b) regular messages*

FIG. 1. *Time for broadcast on Delta over 16 nodes*

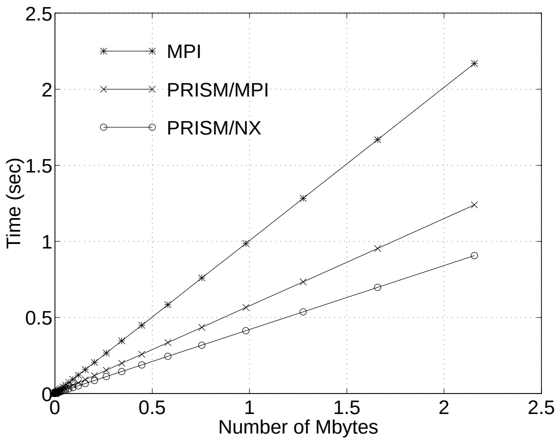
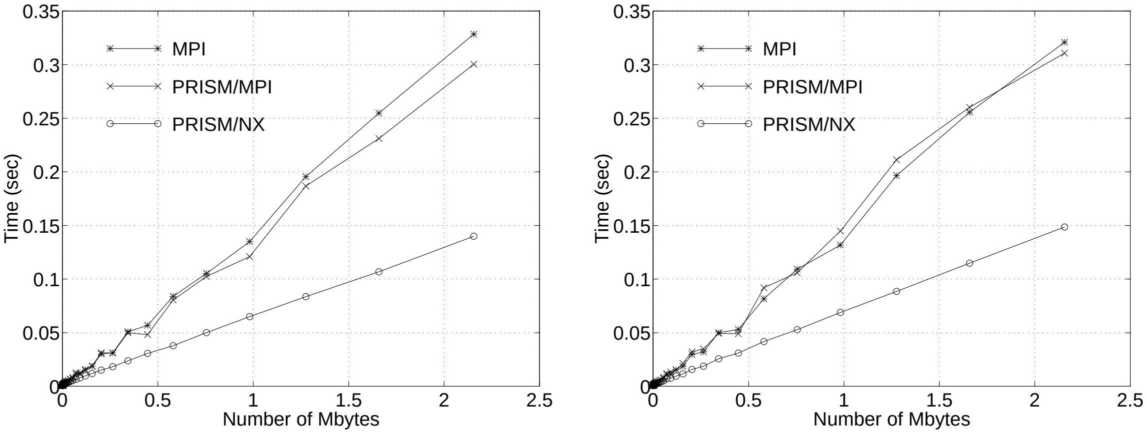


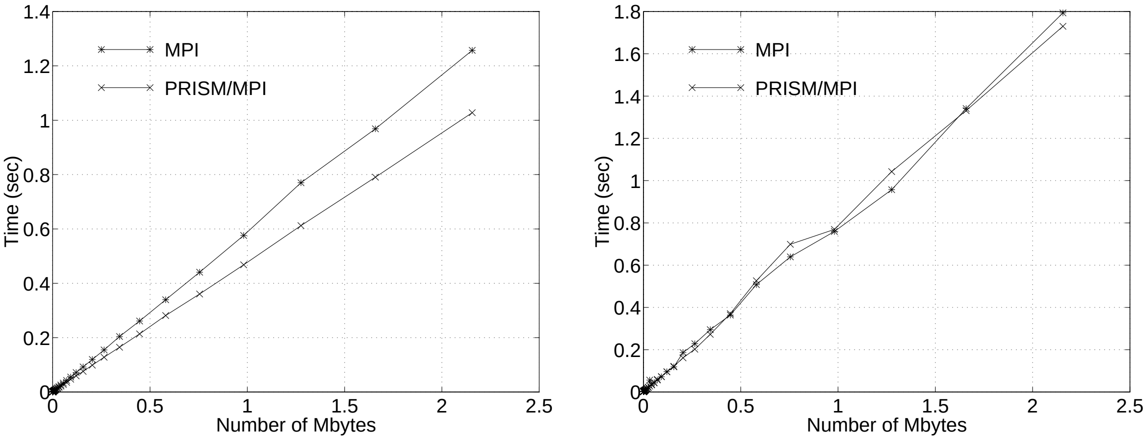
FIG. 2. *Time for broadcast using ready mode on Delta over 15 nodes*

Figure 3a shows the time for broadcasting over 15 nodes on the Paragon using ready mode/forced-type, and Figure 3b shows the same result using regular sends. A similar result is obtained for 16 nodes. Unlike the Delta, the performance of the MPI and PRISM/MPI routines is similar whether or not the number of processors is a power of two. Note, however, that the PRISM/NX code is approximately twice as fast as either MPI version, whether or not ready mode/forced-type messages are used. This is due to the way the Paragon messaging is done, which currently does not allow the ANL/MS version to post receives in an efficient manner. For long messages, PRISM/NX is only about 6% faster using forced-type than using regular messages.



a) ready mode *b) regular messages*

FIG. 3. *Time for broadcast on Paragon over* 15 *nodes*



a) SP1 *b) CM5*

FIG. 4. *Time for Broadcast using ready mode over* 15 *nodes*

Finally, the results for 15 nodes on the SP1 and CM5 are given in Figure 4. On the SP1 and CM5, we do not have an optimized version that uses the vendor's proprietary message passing system, so only MPI results are given. It is seen in Figure 4a that the PRISM/MPI routine is about 18% faster for long messages on the SP1, even though the PRISM routine is using MPI for its message passing. However, it is seen in Figure 4b that the MPI result has the same speed as PRISM/MPI on the CM5 for long messages. The differences seen are attributable to timing variances on the CM5. In addition, the MPI collective routine is almost twice as fast as the PRISM/MPI routine for short messages. The same types of results are obtained on 16 nodes.

4 Conclusions

We have shown that MPI is performing well on many systems. The ANL/MS portable version still has room for improvement, but we feel the performance is good considering that the MPI standard was completed only four months ago. It is also noteworthy that the

ANL/MS MPI runs correctly on many systems. Results for more MPI implementations, which could not be included due to space limitations, will be presented at the conference.

5 References

[1] Auslander, L. & A. Tsao, *On parallelizable eigensolvers*, Adv. Appl. Math. **13** (1992), 253–261.

[2] Beguelin, A., J. J. Dongarra, G. A. Geist, R. Manchek & V. S. Sunderam, *A user's guide to pvm parallel virtual machine*, ORNL/TM-11826, Oak Ridge National Laboratory (July 1991).

[3] Bischof, C. H., S. Huss-Lederman, X. Sun, & A. Tsao, *The PRISM project: infrastructure and algorithms for parallel eigensolvers*, Proceedings, Scalable Parallel Libraries Conference (Starksville, MS, Oct. 6–8, 1993), IEEE, 1993, pp. 123–131, (also PRISM Working Note #12).

[4] Butler, R. & E. Lusk, *User's guide to the p4 parallel programming system*, ANL-92/17, Argonne National Laboratory (October 1992).

[5] Franke, H., P. Hochschild, P. Pattnaik & M. Snir, *An efficient implementation of MPI*, IFIP WG10.3 Working Conference on Program,ing Environments for Massively Parallel Distributed Systems, April 25-29, 1994, (also IBM Research Report: RC 19493(84718) 3/25/94).

[6] Franke, H., P. Hochschild, P. Pattnaik & M. Snir, *MPI-F: An efficient implementation of MPI on IBM-SP1*, 1994 International Conference on Parallel Processing, August 15-19, 1994.

[7] Gropp, W. & E. Lusk, *An abstract device definition to support the implementation of a high-level point-to-point message-passing interface*, MCS-P342-1193, Argonne National Laboratory (1993).

[8] Gropp, William D. & Barry Smith, *Chameleon parallel programming tools users manual*, ANL-93/23, Argonne National Laboratory (March 1993).

[9] Hempel, R., *The ANL/GMD macros (PARMACS) in Fortran for portable parallel parogramming using the message passing programming model - users' guide and reference manual*, GMD (November 1991).

[10] Huss-Lederman, S., E. M. Jacobson, A. Tsao, & G. Zhang, *Optimizing communication primitives on the Intel Touchstone Delta*, Technical Report, Supercomputing Research Center (to appear).

[11] Littlefield, R., *Characterizing and tuning communications performance for real applications*, presentation overheads, Proceedings, First Intel Delta Applications Workshop, CCSF-14-92 (February, 1992), Caltech Concurrent Supercomputing Facilities, Pasadena, California, 1992, pp. 179–190.

[12] Meiko Limited, *Tagged Message Passing & Global Reduction* (1993).

[13] Message Passing Interface Forum, *MPI: A Message-Passing Interface standard*, Computer Science Dept. Technical Report CS-94-230 (April 1994), University of Tennessee, (To appear in the International Journal of Supercomputer Applications, Volume 8, Number 3/4, 1994).

[14] Parasoft Corporation, *Express Version 1.0: A Communication Environment for Parallel Computers*, 1988.

[15] Regnier, G., *Intel Touchstone Delta message passing performance*, Intel Supercomputing Systems Division (August, 1991), preprint.