Comparison of MPI Implementations on a Shared Memory Machine

Brian VanVoorst[1] and Steven Seidel[2]

[1] Honeywell Technology Center, 3660 Technology Drive, Minneapolis, Minn. 55418
brian_vanvoorst@honeywell.com
[2] Dept. of Computer Science, Michigan Technological Univ., Houghton, Michigan 49931 steve@mtu.edu

Abstract. There are several alternative MPI implementations available to parallel application developers. LAM MPI and MPICH are the most common. System vendors also provide their own implementations of MPI. Each version of MPI has options that can be tuned to best fit the characteristics of the application and platform. The parallel application developer needs to know which implementation and options are best suited to the problem and platform at hand. In this study the RTCOMM1 communication benchmark from the Real Time Parallel Benchmark Suite is used to collect performance data on several MPI implementations for a Sun Enterprise 4500. This benchmark provides the data needed to create a refined cost model for each MPI implementation and to produce visualizations of those models. In addition, this benchmark provides best, worst, and typical message passing performance data which is of particular interest to real-time parallel programmers.

1 Introduction

Shared memory platforms can support many different versions of the Message Passing Interface (MPI)[1]. Among the best known MPI implementations are LAM MPI[3] and MPICH[2]. Vendors also provide MPI implementations particularly suited to their platforms. Each implementation has various options for tuning its behavior. This creates several choices for an application developer who is seeking the best possible performance for their application.

The work presented here characterizes several MPI implementations and configurations for the Sun Enterprise 4500. These characterizations are based on data obtained from the RTCOMM1 communication benchmark, part of the Real Time Parallel Benchmark Suite [4, 5]. A refined communication cost model for each implementation is obtained by an iterative process of running RTCOMM1, examining the output, and adjusting the input to focus on the behavioral features revealed by the most recent data. This process was performed for the MPI implementations listed in Table 1.

[2] This work is partially supported by NSF grant MRI-9871133.

Table 1. MPI variations examined

MPI	*Mechanism*	*Option*
Sun	SHMEM	
	SHMEM	MPI_SPIN
	SHMEM	MPI_POLLALL
	SHMEM	MPI_EAGER
LAM	TCP/IP	-O -c2c -nger
	SHMEM	-O -c2c -nger
MPICH	SHMEM	

Several system configuration options can also be varied in order to reveal their impact on message passing performance. The configuration options available on the E4500 include locking processes to processors, disabling interrupts, and even disabling individual processors. The effects of these options were also investigated.

2 Approach

Three MPI implementations are studied: Sun's MPI provided with HPC 3.0, LAM MPI[3], and MPICH[2]. LAM MPI was built in both its default TCP/IP version and in its shared memory version. These two builds of LAM MPI are compared to determine the amount of additional overhead created by the TCP/IP version compared to its shared memory version. By default, MPICH builds a shared memory version. No attempt at building a TCP/IP version of MPICH was made. These four implementations of MPI are the subject of the characterization work presented here. The platform used for this work is an 11-processor Sun Microsystems Enterprise 4500 symmetric multiprocessor with 8GB of memory running Solaris 2.7. The processors are 400MHz Sparc II's with 4MB of cache.

The characterization methodology for this work relies heavily on the use of the RTCOMM1 benchmark. RTCOMM1 takes as input a sequence of message size ranges (*e.g.*, 0-128 bytes, 129-4098 bytes, ...) and for each range produces N sample points. The experiments reported here use $N = 20$. A large number of ping-pong operations (sending a message back and forth between two processes) are timed at each sample point. The exact number of ping-pongs is not specified by the input to the benchmark. Instead, a total run time is specified. The benchmark performs a ping-pong measurement for each sample point in a round-robin fashion until the run time expires. The benchmark terminates only after completing a full round of sample points. This ensures that all message size ranges are measured an equal number of times and that any interruption of the benchmark (by, for example, an increase in background load) will not significantly bias the measurement of any one sample point.

For each sample point RTCOMM1 records the fastest (best), slowest (worst), and typical (median) time to complete a ping-pong. At the completion of the

benchmark RTCOMM1 fits a line to the typical points of each message size range. This line is the communication cost model for that range of message sizes. RTCOMM1 provides as output these cost models and a series of data files suitable for plotting.

The initial approach to the characterization of each MPI implementation is to oversample with short message ranges and a dense set of sample points. This provides a fine-grained picture of point-to-point communication performance. These measurements reveal interesting regions in the graph of the performance data. It is usually apparent that there are certain message size ranges for which different underlying protocols, buffering schemes, *etc.* are used. Transitions in the graph at the boundaries of these ranges illustrate changes in the performance of the MPI implementation. Based on these observations, the input to the benchmark is adjusted so that the selected ranges match the transition points of the oversampled runs. A few iterations of this approach produces an accurate cost model for each MPI configuration.

3 Results

Due to limited space, only graphs of the most interesting characteristics of the MPI implementations are presented. These features appear at a variety of scales. Those that have the most direct impact on performance are discussed here.

It is important to remember that each point on a graph represents thousands of individually timed messages. When a "best" point takes slightly longer than its neighboring best points it is not due to chance mis-measurement. It is the result of some artifact in the system that did not allow that message to be transmitted faster. Not all such abnormalities can be explained, but they can be measured and their impact on performance can be revealed. All data points shown are actual measurements of point-to-point communication, not averages, computed by halving the best, typical, or worst observed ping-pong measurements.

For all three message passing libraries the typical times are often the same as (or very close to) the best times. This means that out of thousands of trials the median time is usually the same as the best time. Therefore, application developers can be confident that they will usually receive the best possible message passing performance the system has to offer. However, poorer performance sometimes occurs. This is captured by the "worst" observed points. These points often differ by a constant from the best observed times. This constant may be the cost of servicing one interrupt, which might happen only infrequently.

Messages were occasionally observed to be slowed down by orders of magnitude, up to $1/10^{th}$ of a second. This phenomenon can be reproduced by binding the benchmark processes to specific processors, disabling interrupts on those processors, and disabling all other processors except one. This caused many messages delays ranging from 0.02 to 0.1 seconds at consistently spaced intervals of 0.01 seconds. It is unclear why this particular combination of circumstances caused this delay.

3.1 Platform Configuration

Experiments showed that binding a processes to a processor fostered consistent performance. The `processor_bind()` system call prevents the operating system from migrating processes among processors. Under these conditions it appeared that the operating system will not schedule these processors for other work if other processors are available. Using the `psradm` command to disable I/O interrupts on these processors further reduced the possibility of these processors being interrupted while running the benchmark. Experience also indicated that it was necessary to leave more than one processor available for servicing interrupts.

The results presented here were collected from two benchmarking tasks bound to processors 0 and 1 on which interrupts were disabled. The remaining nine processors were available for other purposes but no other user jobs were running on the machine.

3.2 Sun's HPC 3.0 MPI

Sun's MPI implementation delivered the fastest overall point-to-point message passing. However, Sun's MPI was the least consistent and hardest to model for larger-sized messages. Figure 1 shows a plot of messages sizes between the ranges of 210KB and 240KB. No explanation can be offered for the illustrated oscillations in message passing times. While the variance is small (5%), it is large relative to the execution time. This shows that for certain message sizes, a message that is a few bytes longer may be transmitted in less time (as much as 50 microseconds) than the shorter message. This effect is reproducible and starts to occur for messages longer than 64KB. A second observable trend (not illustrated here) is that the difference between the worst and best points increases with message length. This is probably due to an increased chance of being interrupted multiple times while sending a longer message. The cost model for Sun MPI is given in Table 2. Due to the variance in measurements seen in Figure 1 it is not possible to present a precise cost model for messages longer than 64KB.

Table 2. Sun MPI cost model (*Imprecise due to large variance)

Message size (bytes)	Latency (μsec)	Bandwidth (MB/sec)
0 - 256	6	41.53
256 - 512	10	236.7
512- 1K	9	158.2
1K -16K	11	182.0
16K - 32K	29*	197.9*
32K - 1M	35*	208.6*
1M - 2M	277*	219.7*
2M - 4M	594*	225.1*

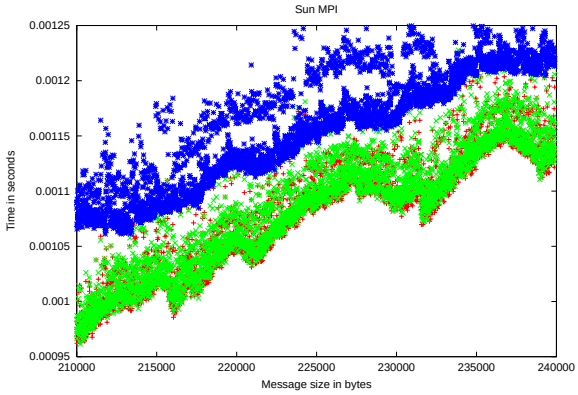

Fig. 1. Sun's MPI performance varies for large message sizes

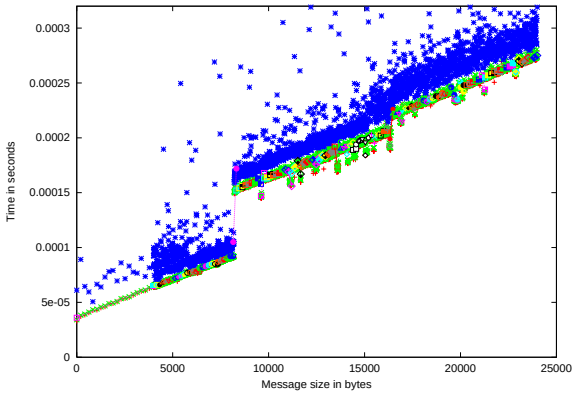

Fig. 2. Oversamples LAM MPI performance

3.3 LAM Shared Memory MPI

The most distinctive feature of LAM MPI is a large jump in latency for messages of length 8KB, as shown in Figure 2. The magnitude of this increase shows that it takes almost twice as long to transmit a message of length 8193 as it does to transmit a message of 8192 bytes. For longer messages LAM message passing costs are modeled well by a straight line, as given in Table 3. Both the shared memory version of LAM and the TCP/IP version of LAM were built and tested in these experiments but the measurements were the same in both cases. Because the bandwidth is so high in each case it must be that the TCP/IP version is making use of shared memory.

Table 3. LAM cost model

Message size (bytes)	Latency (μsec)	Bandwidth (MB/sec)
0-256	33	44.48
256 - 8K	35	145.9
8K - 1M	108	141.9
1M - 4M	108	141.7

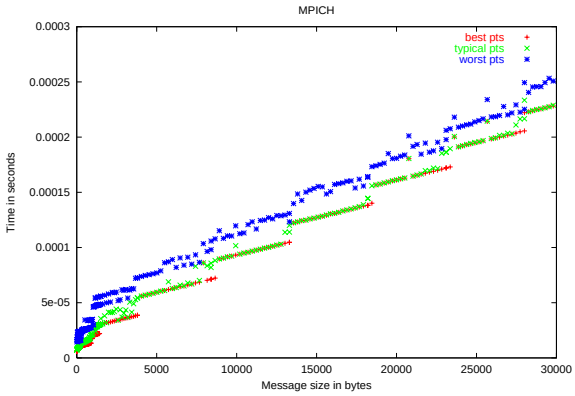

Fig. 3. MPICH cost model is a step function

3.4 MPICH

The performance of MPICH is best characterized by a step function. Figure 3 shows the observed message passing times for MPICH for messages of lengths 0 to 30,000 bytes. The interval of the step shown in Figure 3 is about 4900 bytes and it varies slightly as the message size grows. It then changes to an interval of about 9800 bytes when the message size is greater than 130,000 bytes. This interval also changes slightly as message size grows. The cause for this step function and the variation of interval size is not known but it might be a side effect of padding or buffer allocation and usage. The cost model for MPICH is given in Table 4. For messages longer than 100KB MPICH exhibits two "levels" of message passing times for each message length. Figure 4 shows message passing times for messages between the sizes of 200,000 and 210,000 bytes. Note that about a third of the time message passing times are greater by a fixed amount. This behavior is reproducible but no explanation can be offered here.

4 Conclusions

Sun MPI offers the best performance of the three MPI implementations. Figure 5 summarizes the costs of passing long message using Sun MPI, MPICH, and LAM. The second-order performance characteristics of these message passing interfaces are illustrated in Figures 1-4. Figure 1 shows that for long messages Sun MPI

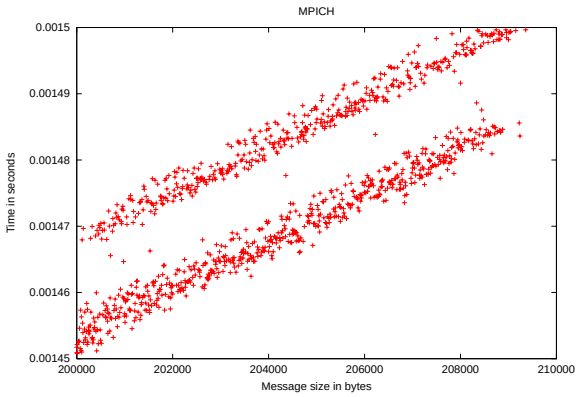

Fig. 4. MPICH bimodal cost behavior

Table 4. MPICH cost model

Message size (bytes)	Latency (μsec)	Bandwidth (MB/sec)
0 - 256	7	48.2
256 - 512	12	195.4
512 - 1K	6	56.5
1K - 130K	$\frac{Size-1K}{4900}$ * 5	282.3
130K - 4M	$\frac{Size-130000}{9800}$* 5+400	238.7

exhibits large cost fluctuations. Figure 2 shows that with LAM, messages longer than 8KB have a start up cost three times that of messages shorter than 8KB and that LAM performance is best modeled by one cost function for messages shorter than 8KB and by another for messages longer than 8KB. MPICH is best characterized by a step function whose latency increases with message length, as shown in Figure 3. Figure 4 illustrates MPICH's bimodal cost behavior. The best platform configuration across all MPI implementations required locking processes to processors and disabling interrupts on those processors. These steps helped to ensure that processors remained dedicated to the application.

MPI implementations on the same machine, using the same shared memory message transport mechanism, have very different performance characteristics. The results presented here illustrate significant differences among cost models, scaling behavior, worst-case performance, and other performance characteristics. These differences stem from implementation decisions made by interface developers. LAM and MPICH are portable MPI implementations that are not tuned for specific platforms. The native implementation has a clear advantage in this case. It is also clear that no single implementation of MPI is best for all applications. This suggests that similar studies should be done for other platforms.

It has been shown here that RTCOMM1 can be used to characterize MPI implementations. The communication cost model of a message passing interface and hardware platform is usually described as a linear function determined by

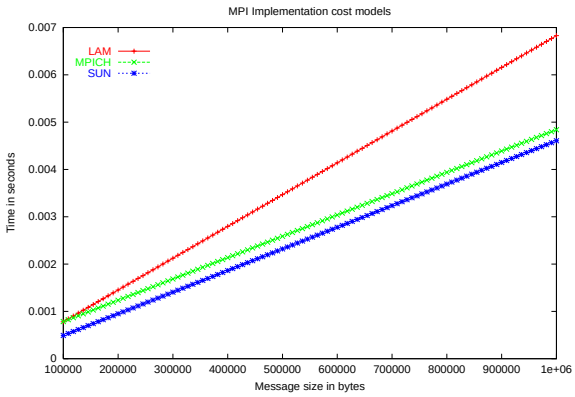

Fig. 5. Comparative message passing performance

a measured startup cost (latency) and bandwidth. RTCOMM1 was used here to show that this is sometimes an oversimplification. This work also demonstrated an approach for using RTCOMM1 to identify and illustrate performance differences between MPI implementations. While this approach does not reveal the causes underlying those differences, the experimental data does admit the construction of more accurate cost models. In addition, RTCOMM1 provides insight into best and worst case message passing performance which is useful for real-time software development.

References

[1] 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.

[2] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`, *a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[3] A. Lumsdaine, *et al. LAM MPI Home page.* `http://www.mpi.nd.edu/lam/`. University of Notre Dame.

[4] B. VanVoorst, R. Jha, S. Ponnuswammy, C. Nanvati, and L. Pires. DARPA Real Time Parallel Benchmarks: Final report. Technical Report (C013) - Contract Number F30602-94-C-0084, Rome Laboratory, USAF, 1998.

[5] B. VanVoorst, S. Ponnuswammy, R. Jha, and L. Pires. DARPA Real Time Parallel Benchmarks: Low-level benchmark specifications. Technical Report (C006) - Contract Number F30602-94-C-0084, Rome Laboratory, USAF, 1998.