LA-UR 96-4726

CONF-970729--1

Title: Shared Memory vs. Message Passing: the COMOPS Benchmark Experiment

Author(s): Yong Luo

Submitted to: ICS '97
Vienna, Austria
July 7-11, 1997

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Los Alamos
National Laboratory

HH

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. The Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

December 9, 1996

Shared Memory vs. Message Passing: the COMOPS Benchmark Experiment

Yong Luo

CIC-19, Mail Stop B256

Los Alamos National Laboratory

Los Alamos, NM 87545, U.S.A.

Email: yongl@lanl.gov, Fax: (505) 667-1126

Abstract

This paper presents the comparison of the COMOPS benchmark performance in MPI and shared memory on three different shared memory platforms: the DEC AlphaServer 8400/300, the SGI Power Challenge, and the HP-Convex Exemplar SPP1600. The paper also qualitatively analyzes the obtained performance data based on an understanding of the corresponding architecture and the MPI implementations. Some conclustions are made for the inter-processor communication performance on these three shared memory platforms.

Introduction

Parallel computing on shared memory multi-processors has become an effective method to solve large scale scientific and engineering computational problems. Both MPI and shared memory are available for data communication between processors on shared memory platforms. Normally, performing inter-processor data communication by copying data into and out of an intermediate shared buffer seems natural on a shared memory platform. However, some vendors have recently claimed that their customized MPI implementations performed better than the corresponding shared memory protocol on their shared memory platforms even though the MPI protocol was originally designed for distributed memory multi-processor systems. This situation makes it hard for users to choose the best tool for inter-processor communication on those shared memory platforms on which both MPI and shared memory protocols are available. In order to clarify this confusion, a comparison experiment was conducted to illustrate the communication performance for the COMOPS operations on major shared memory platforms. This paper presents the experimental results and presents some qualitative analyses to interpret the results.

This paper has four sections. In the first section, the architectures of three shared memory platforms are briefly described. The implementation details of the experiment are described in the second section. The second section also discusses the shared memory simulation of those communication patterns defined in the COMOPS benchmark set. The third section presents the data and analyses. It graphically exhibits the collected communication performance data and qualitatively interprets the performance behavior based on an understanding of underlying architectures. In the final section, some conclusions and recommendations are made regarding the interprocessor communication performance on the three shared memory platforms.

Architectures

Currently there are two types of shared memory connections for multi-processor systems. One is the bus-connected shared memory system as illustrated in Figure 1. The DEC AlphaServer 8400/300 and the SGI Power Challenge have this type of architecture.

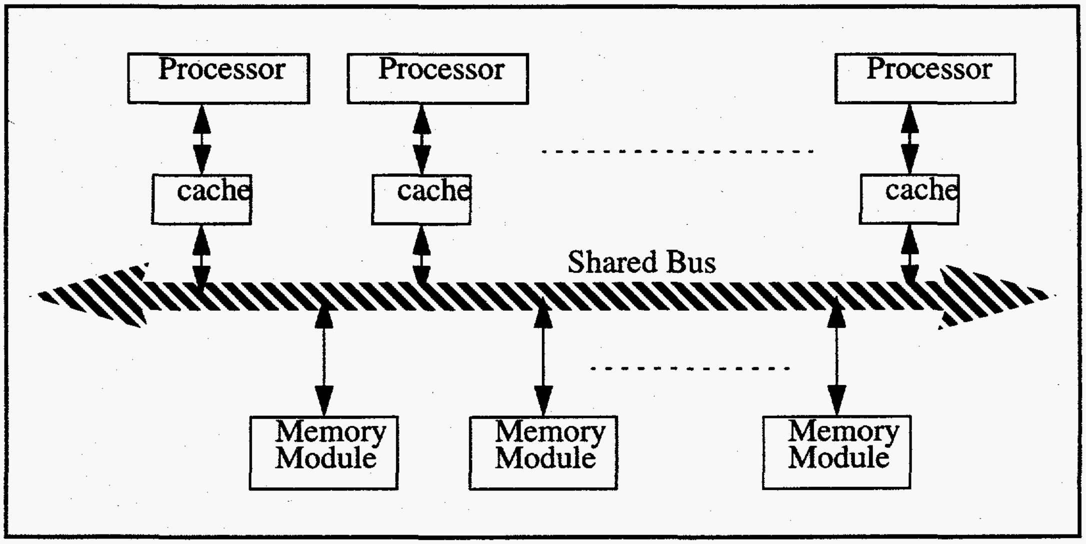

Figure 1. Bus-connected shared memory multiprocessors

In this type of system every processor has equal access to the entire memory system through the same bus. Another type of shared memory multi-processor connection architecture is the crossbar switch. This crossbar connection is a typical connection mechanism within one hypernode of many distributed shared memory (DSM) systems such as HP-Convex Exemplar and NEC SX-4. The Exemplar SPP architecture is shown in Figure 2. The Convex machine we have access to (courtesy of Convex) is a one-hypernode 8-processor machine. The inter-hypernode connection is irrelevant to this experiment and this paper focuses on the intra-hypernode structure only.

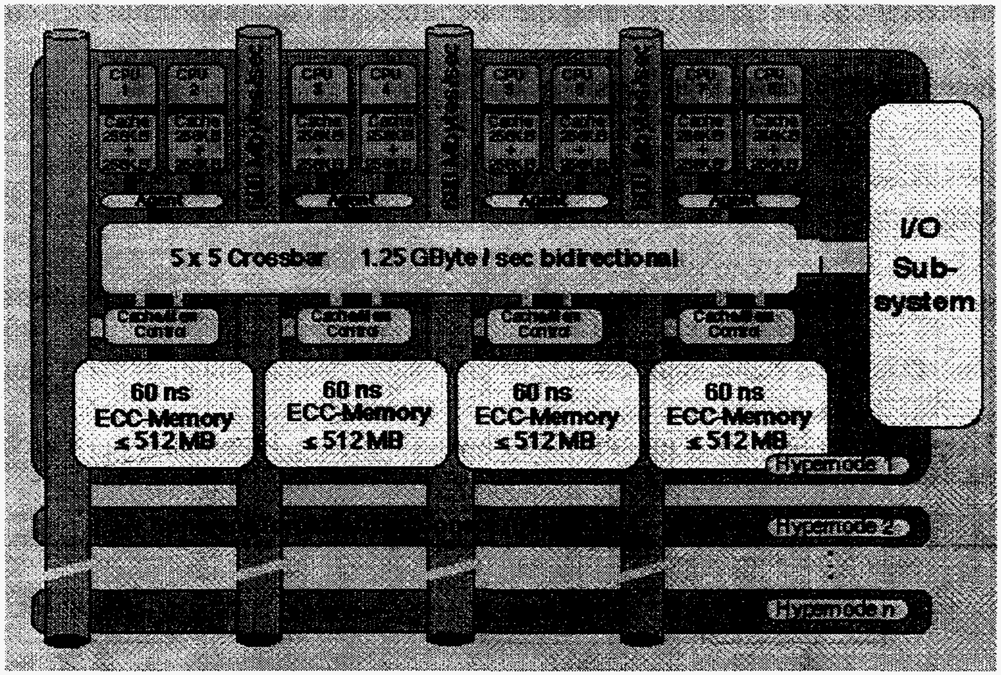

Figure 2. Convex Exemplar Hypernode Structure

The memory access pattern and the physical distance between two processors are different in bus-connected and distributed shared memory systems. In a bus-connected shared memory structure, the memory access for each processor is uniform. But in a distributed shared memory structure, the memory access is non-uniform. This structure is called a NUMA (Non-Uniform Memory Access) architecture. Also, the inter-processor communication in bus-connected shared memory systems is homogeneous and every processor is equi-distant to any other processor in the same system. On the other hand, in a NUMA system such as Convex SPP, a processor always has some neighbors electrically closer than the others in the system. As illustrated in Figure 2, even though the memory access is still uniform within one hypernode of the SPP1600, each processor is electrically closer to the one shared with the same agent because it does not need to go through the crossbar switch for the inter-processor communication.

In this experiment, none of the three shared memory machines has a physical implementation for CPU-private or thread-private memory. In a bus-connected multiprocessor system, such as the SGI Power Challenge and the DEC AlphaServer 8400/300 (nickname Turbolaser), the memory system is purely homogeneous. Therefore, there is no physical distinction between a logically-private memory space and a logically-shared memory space. For the NUMA system SPP1600, although it is a DSM system, its CPU-private or thread-private memory is not physically implemented (HP-Convex, 1994). Instead, the operating system partitions hypernode-private memory (memory modules within one hypernode) used as CPU-private memory for each of the processors in the hypernode. The reason for this is that implementation of a physical CPU-private memory would not result in substantially lower CPU-to-memory latency, and the latency from a processor to hypernode-private memory would be increased (HP-Convex, 1994).

The Experimental Method

The direct objective of this experiment is to clarify the difference in the performance of inter-processor communication between the shared memory protocol and the message passing protocol on a shared memory platform. To achieve this goal, the common inter-processor communication operations specified in the LANL COMOPS benchmark set are used to perform the comparison. The point-to-point communication operation actually used in this experiment is ping-pong. The tested collective operations include broadcast, reduction, gather, and scatter.

The COMOPS benchmark set is designed to measure the performance of inter-processor point-to-point and collective communication in MPI. It measures the communication bandwidth and message transfer time for different message sizes. The set includes ping-pong, broadcast, reduction, and gather/scatter operations. The MPI performance measurement can be directly performed on the three platforms with the corresponding best available MPI implementation. Both SGI and HP-Convex have their own customized MPI implementations on their shared memory platforms. Although the current version of MPI implementation on our DEC AlphaServer 8400/300 Turbolaser is a public-domain MPICH version, according to the information from DEC, this MPICH implementation performs no worse than the DEC customized version MPI within one shared memory multi-processor box. The main effort of this experiment is to write a shared memory version of the COMOPS benchmark set. The shared memory version of these communication operations is illustrated in the following pseudo-code.

```
Ping-pong:
    call timer
      do ntimes
   if (my_thread .eq. 0) then
    shared_temp=private_send     !! Thread 0 sends out message
   endif
   barrier                  !! synchronization
   if (my_thread .eq. 1) then
    private_val=shared_tmp      !! Thread 1 receives the message
    shared_tmp2=private_recv     !! Thread 1 sends out the message
   endif
   barrier                  !! synchronization
   if (my_thread .eq. 0) then
    private_val=shared_tmp2      !!Thread 0 receives back the message
   endif
      enddo
    call timer

Broadcast:
    call timer
      do ntimes
   if (my_thread .eq. 0) then
    shared_temp=privated_send    !! Thread 0 sends out message
   endif
   barrier                  !! synchronization
   if (my_thread .ne. 0) then   !! Other threads receives the
    private_recv=shared_tmp      !! message simultaneously
   endif
   barrier                  !! synchronization
      enddo
    call timer
```

Reduction (global max):

```
 call timer
  do ntimes
 critical section
  shared_tmp=max(shared_tmp, private_send)
 end critical section
   barrier             !! synchronization
if (my_thread .eq. 0) then    !! Thread 0 collects the final
 private_recv=shared_tmp      !! result
endif
  enddo
 call timer
```

Gather:

```
 call timer
  do ntimes
  shared_tmp(j+my_thread*N_size)=private_send(j)
   barrier             !! synchronization
if (my_thread .eq. 0) then    !! Thread 0 collects the final
 private_recv=shared_tmp      !! result
endif
  enddo
 call timer
```

Scatter:

```
 call timer
  do ntimes
if (my_thread .eq. 0) then
 shared_tmp=private_send      !! Thread 0 sends out the message
endif
   barrier             !! synchronization
 private_recv(j)=shared_tmp(j+my_thread*N_size)
   barrier             !! synchronization
  enddo
 call timer
```

This experiment actually involves two versions of shared memory codes because of the different shared memory programming environments. The shared memory programming environment on both the DEC AlphaServer and SGI Power Challenge systems is compatible with PCF (Parallel Computing Forum) standard. Therefore, only one version of code is needed for these two machines. The Convex shared memory programming feature in Fortran is slightly different. In particular, in the operation of ping-pong, a lock-and-wait mechanism, instead of the general synchronization barrier, can be used for the synchronization between Processor 0 and Processor 1.

As shown in the pseudocode list, only one pair of processors participate in the operation of ping-pong, regardless of the total number of processors involved. The collective communication operations involves all the processors in the run. The shared memory version accomplishes the same operations performed in the original MPI version of the COMOPS benchmark.

Performance Data and Analysis

The original MPI COMOPS benchmark set and the equivalent multi-thread shared memory version have been run on three platforms outlined in Table 1 (SGI, 1995, DEC, 1995 & Reed, 1996). On both SGI and Convex machines, vendor's customized version of

MPI are used in this experiment. On the DEC Alpha machine, a public-domain MPI implementation (MPICH) is used.

TABLE 1.**Three Tested Shared Memory System Configurations**

	SGI Power Challenge	DEC Turbolaser	Convex SPP1600
CPU/clock	8 * MIPS R10K/ 194MHz	10 * Alpha 21164/ 300MHz	8 * HP 7200 / 120MHz
Data Cache	L1: 32KB L2: 2MB	L1: 8KB L2: 96KB L3: 4MB	L1: 1MB (plus 2KB on-chip cache)
Memory	2304MB 1-way interleaved	4GB 8-way interleaved	1Gb 4-way interleaved
Peak Connecting Bandwidth	1.2GB/sec	1.6GB/sec	1.25GB/sec

The collected performance data are illustrated in Figures 3 through 17. Figures 3 through 5, Figure 7, and Figure 8 exhibit the cross-platform bandwidth comparison and the comparison between the shared memory communication protocol as well as the message passing communication protocol. These performance data are all obtained using four processors with different message sizes. It is clear that the performance of the SGI MPI is generally superior to the other ones (except for pingpong performance). The SGI MPI is also better than its corresponding shared memory performance on all 5 communication operations (ping-pong, broadcast, reduction, gather, and scatter).

More specifically, on the SGI Power Challenge, MPI is about three times faster than shared memory for the performance ping-pong. The broadcast performance on this SGI shared memory machine is about the same for MPI and shared memory. Scatter operations in the SGI MPI are nearly 10 times faster than shared memory for medium and big message sizes. As for gather operations, MPI bandwidth is nearly five times higher than shared memory bandwidth for medium message size. For a message size of 800KB, this MPI performance still holds at the level of twice as fast as the shared memory.

The DEC AlphaServer 8400/300 has comparable MPI and shared memory performance for the ping-pong operation. But for all the tested collective operations (broadcast, reduction, gather, and scatter), its shared memory bandwidth is considerably higher than the MPI bandwidth.

On the Convex Exemplar SPP1600, the Convex-customized MPI performs eight times faster than its shared memory does for the ping-pong operation. The Convex MPI is also the best one in terms of pingpong performance. For the other four collective operations, the performance of MPI is just slightly better than that of shared memory method.

Figure 6 demonstrates the ping-pong round trip transfer time for small message sizes (8 Bytes to 80 Bytes). This performance typically reflects the communication latency. It is clear that the shared memory method on the DEC AlphaServer8400 has the lowest ping-pong latency. In Figures 9 through 17, the performance behaviors for ping-pong, broadcast, and reduction are respectively shown on each platform for a fixed message size (800KB) with different number of processors. It should be noted that the bandwidth calculation of ping-pong in COMOPS is what some people called "ping-pong rate =

message_size / round_trip_time". So, it's only half of the "one-way" ping-pong bandwidth as other benchmark reported.

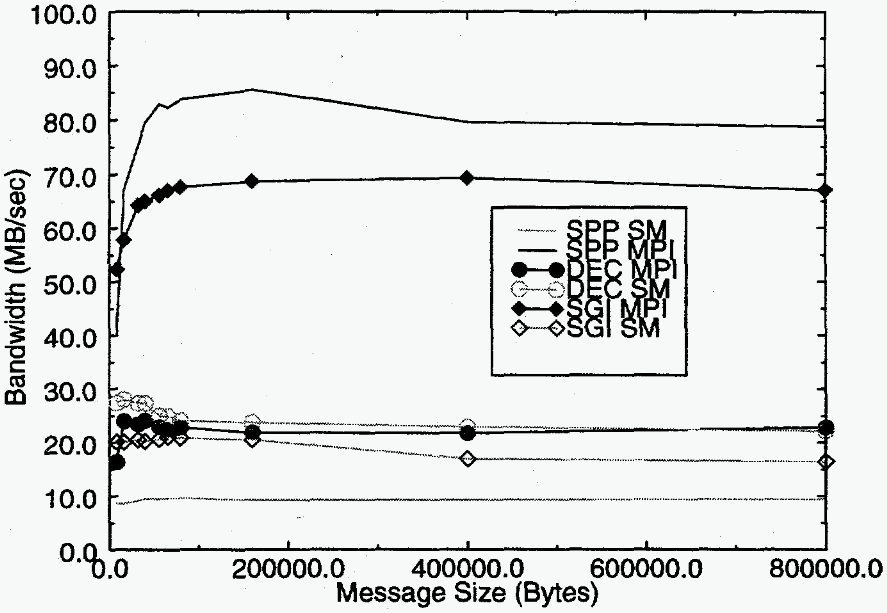

Figure 3. Pingpong Rate

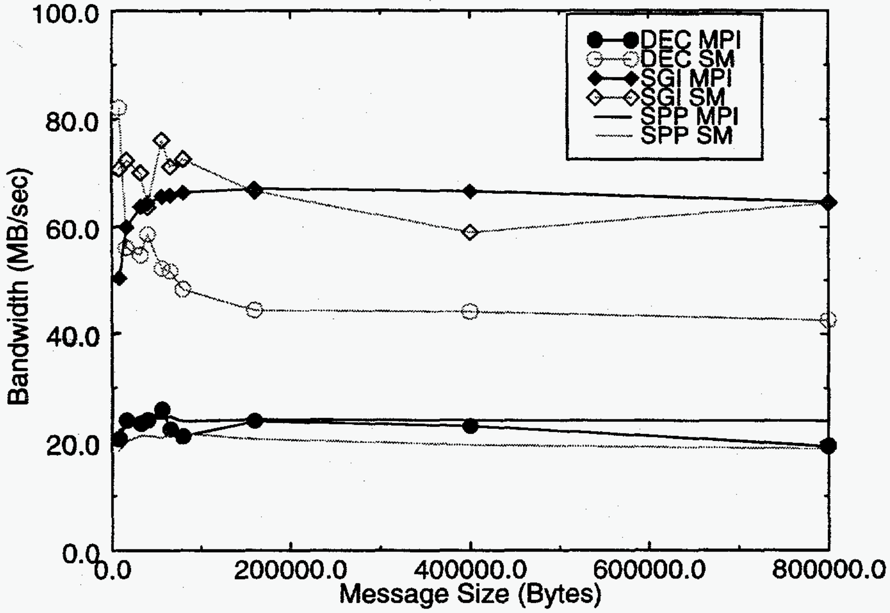

Figure 4. Broadcast Bandwdith

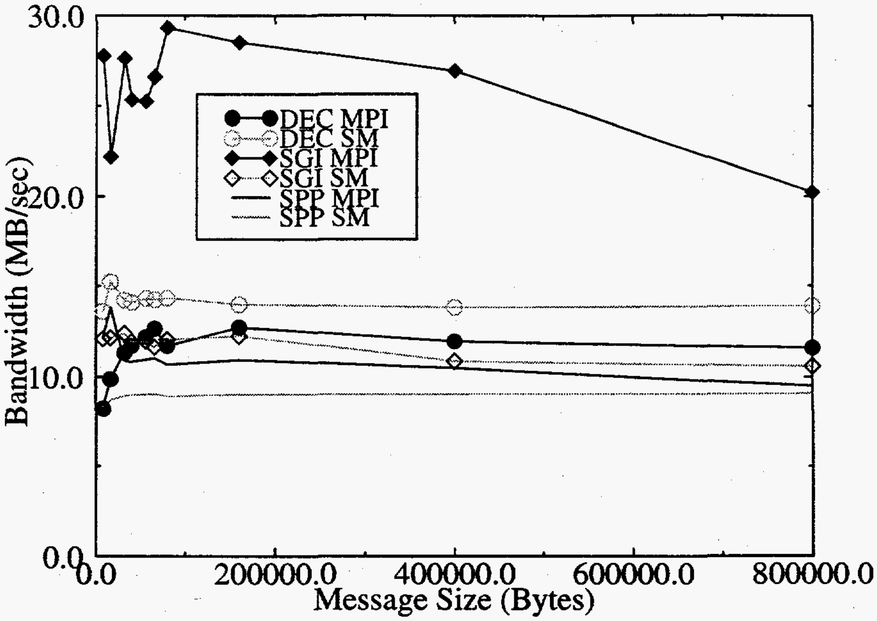

Figure 5. Reduction Bandwdith

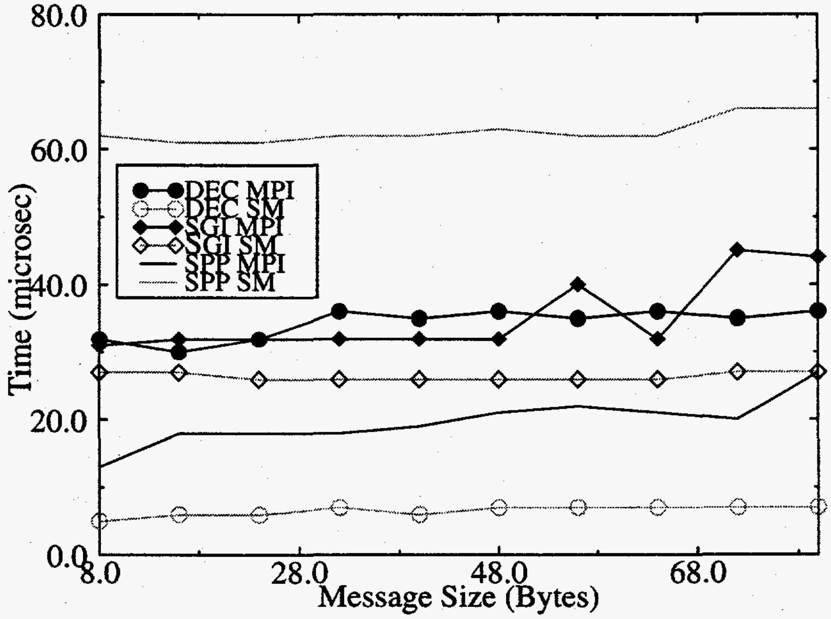

Figure 6. Small Message ping-pong Time

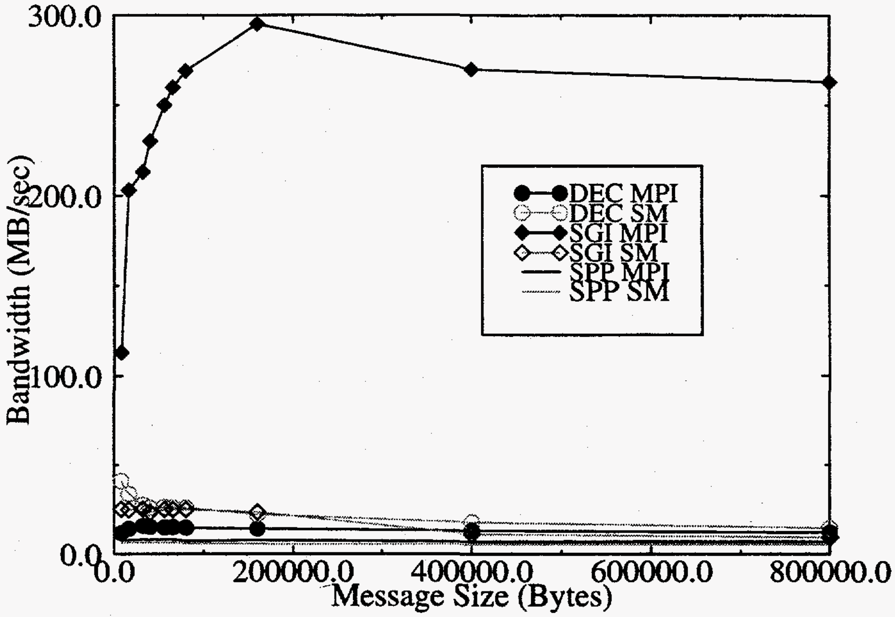

Figure 7. Global Scatter Bandwidth

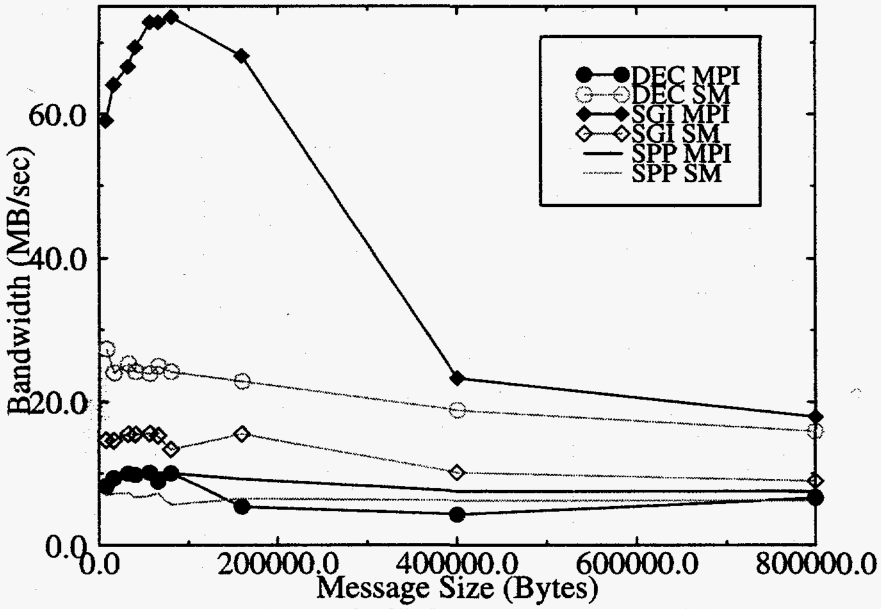

Figure 8. Global scatter Bandwidth

Figure 7 and Figure 8 reflect a big difference between the gather and the scatter bandwidth in the SGI MPI. According to Eric Salo (1996), a SGI MPI expert, for scatter operations, the root processor essentially sends a pointer and a length of the targeted data block to each of the slave processors, which then copy the data in parallel. This turns out to be the situation in which every slave processor directly reads the corresponding block of data from the space owned by the root processor. Since in scatter operations, every slave processor reads a different block of data, virtually no memory conflicts exist, and all processors can read the data at full bandwidth. But for gather operations, the situation is reversed. The root processor has to move the data from different locations all by itself. So, the gather bandwidth is limited by this implementation at the level of about 70MB/sec.

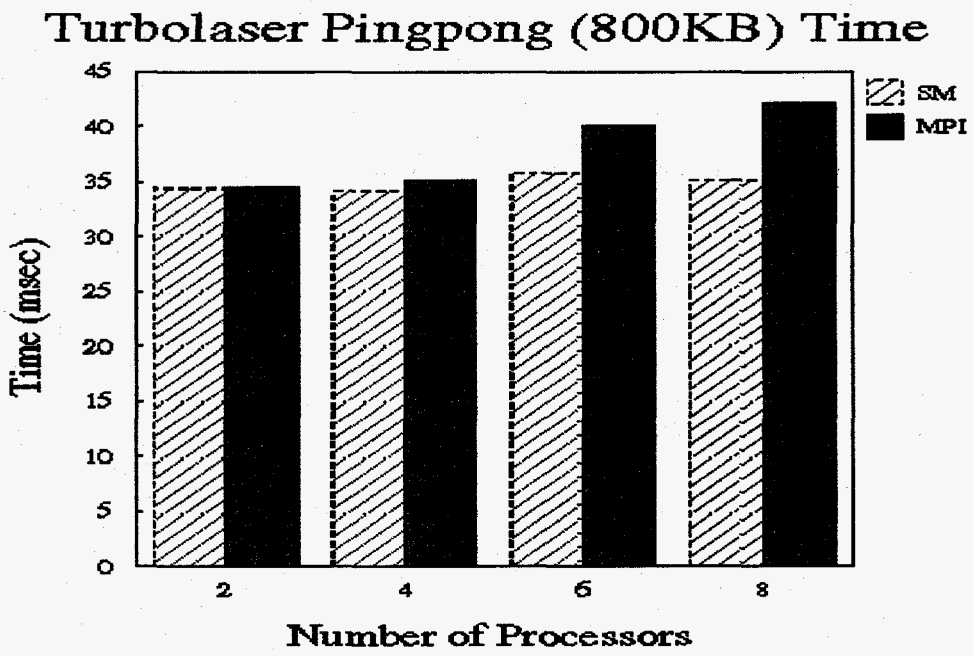

Figure 9. Turbolaser ping-pong Time

Now, based on an understanding of architectures and the underlying MPI implementations, the qualitative performance analysis of ping-pong, broadcast, and reduction operations on each platform is presented here. Figure 9 shows the ping-pong time on the DEC AlphaServer for a fixed message size (800KB) with different number of processors involved. On this DEC machine, MPI is built on top of its shared memory communication protocol. Therefore, MPI performance is always slightly worse than shared memory because of the overhead involved in the MPI implementation. Also, MPI processes seem to be "heavy". Although only two processors participate in the ping-pong operation, the time slightly grows up when the number of MPI processes increases. This is probably due to the interruption from the operating system and the other MPI processes, which are supposed to be idle. On the other hand, the time for the shared memory ping-pong operation remains constant, regardless of the number of processors in the run. This is because the cache coherence caused by invalidating the shared cache line on each processor is performed by broadcasting the message on the bus, instead of sending it to each processor separately.

The broadcast performance on the DEC AlphaServer (Figure 10) is easy to understand. The increase of the shared memory broadcast time with more processors is caused by the increasing queue length of the slave processors. In MPI, the synchronization cost causes the broadcast time to increase more significantly with more processors. The same situation holds for reduction (Figure 11). However, because the shared memory reduction involves a critical section (as listed in the pseudocode), the reduction time increases more as more processors are waiting to enter the critical section.

Similarly, the ping-pong operation has a flat performance on the SGI Power Challenge (Figure 12). The difference from the situation of the DEC AlphaServer is that the MPI ping-pong time does not grow up with more processors. It looks like the MPI processes are "light" on the SGI Power Challenge because the OS interruption does not steal the effective bandwidth even if all processors are in the run. The SGI implementation of MPI is based on the global memory copy function **Bcopy()** (Salo, 1996). Thus, the ping-pong operation is accomplished by directly copying data from the space owned by the source processor to the destination processor, without going through an intermediate

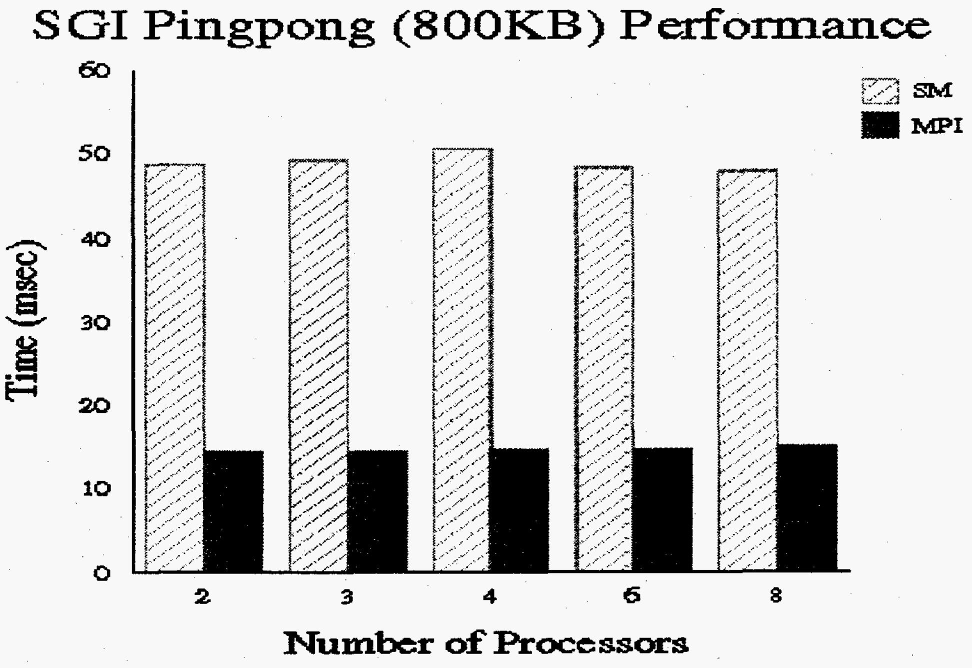

Figure 12. SGI ping-pong Time

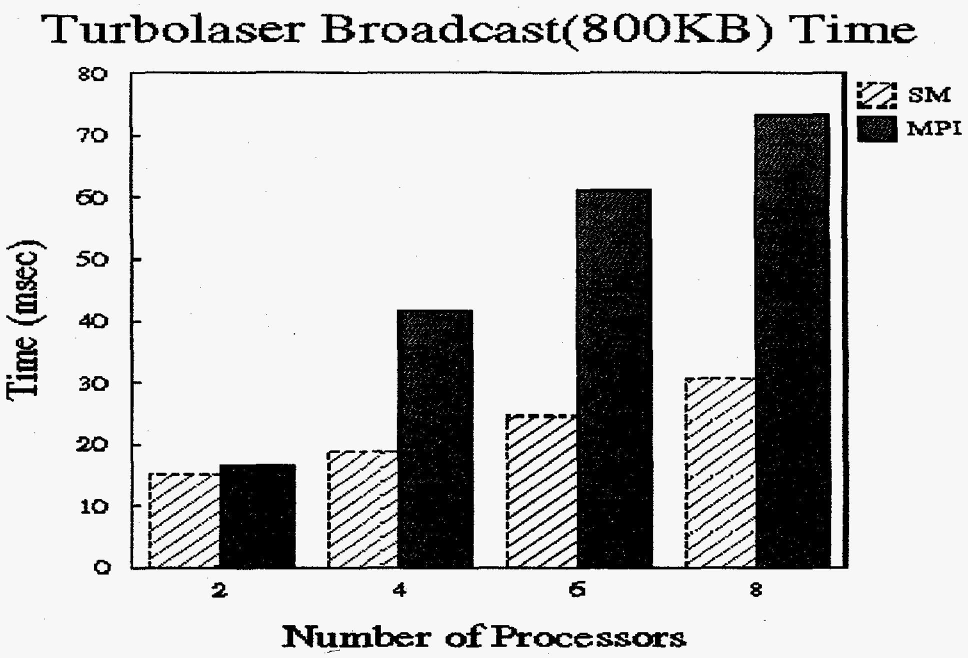

Figure 10. Turbolaser Broadcast Time

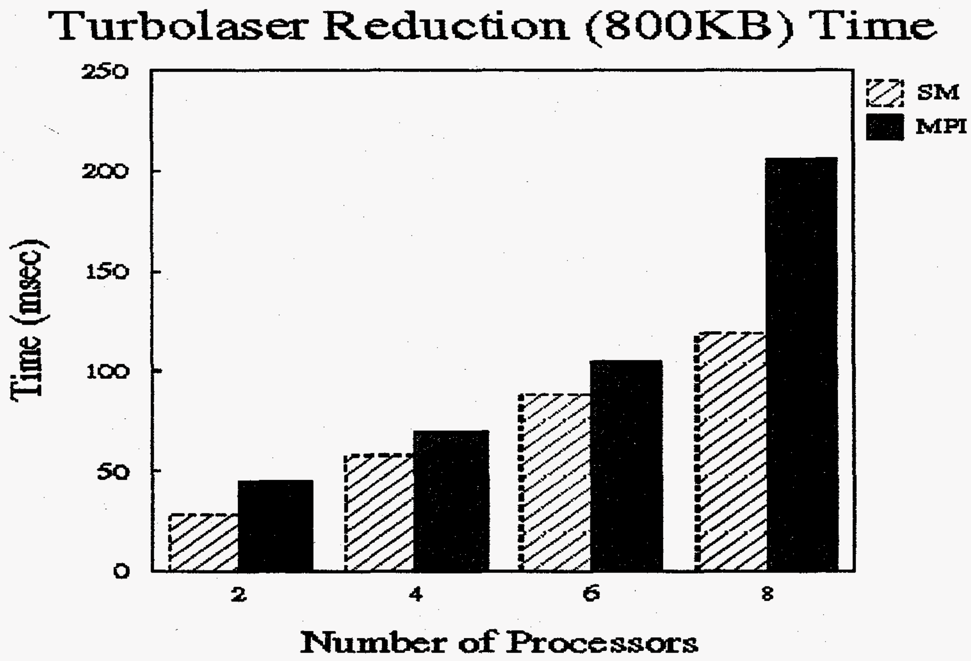

Figure 11. Turbolaser Reduction Time

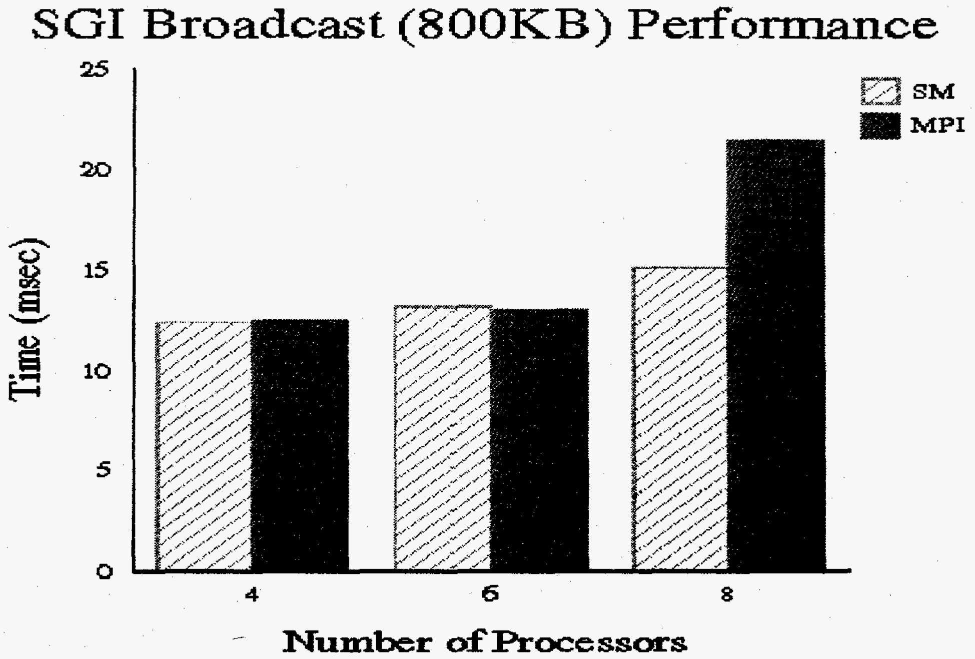

Figure 13. SGI Broadcast Time

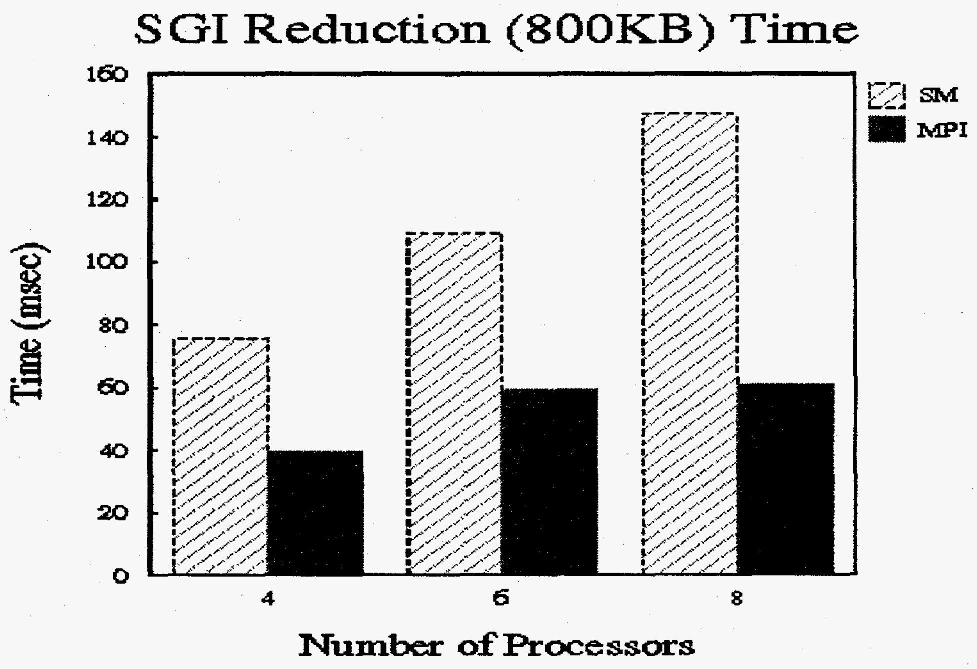

Figure 14. SGI Reduction Time

shared space (Gropp, Lusk, Doss & Skjellum, 1996). Therefore, the shared memory scheme, which uses an intermediate shared space as an interim, takes more than twice as long as MPI does.

The performance of shared memory broadcast and reduction on the SGI machine (Figure 13 and 14) is similar to what is observed on the DEC AlphaServer because of the identical architecture and the same version of shared memory code. The time for broadcast grows up with more processors because of the increasing queue length for reading the shared space. For reduction, the cost from the critical section increases with more processors involved. The MPI performance behaviors for broadcast and reduction on the SGI Power Challenge are interesting. In fact, the MPI performance illustrated in Figure 13 and 14 reflect the underlying implementation of the SGI MPI. The MPI operation for broadcast is implemented as a fan-out tree on the top of the **Bcopy()** point-to-point mechanism (Salo, 1996). For reduction operations, it is in the reversed order as a fan-in tree. Both of them have some parallelism as each pair of processors can perform fan-in or fan-out independently. Since the algorithm of fan-in/fan-out tree requires a synchronization at each tree-fork/join stage, the cost of broadcast/reduction will grow up with more fork/join synchronizations as more processors participate into the operation. Therefore, the time for reduction on eight processors is nearly the same as that for six processors because they both involve the same number of join synchronization stages. The big growth in the time for broadcast on eight processors (Figure 13) in fact is caused by the synchronization at the completion of broadcast. With all the processors in the system being synchronized at certain point, the OS overhead can be significant. On the other hand, there is no need for such a synchronization in reduction.

The ping-pong performance on the Convex SPP1600 (Figure 15) is very similar to that on the SGI Power Challenge. From the phenomenon that the MPI takes nearly half time of what the shared memory scheme takes to perform the ping-pong operation, it is reasonable to anticipate that the MPI implementation on the SPP1600 may be also based on the direct memory copy, instead of going through an intermediate shared space (Gropp, *et al,* 1996). Also, some special manipulations must have done to achieve nearly 8 times faster pingpong speed in the Convex implementation of MPI.

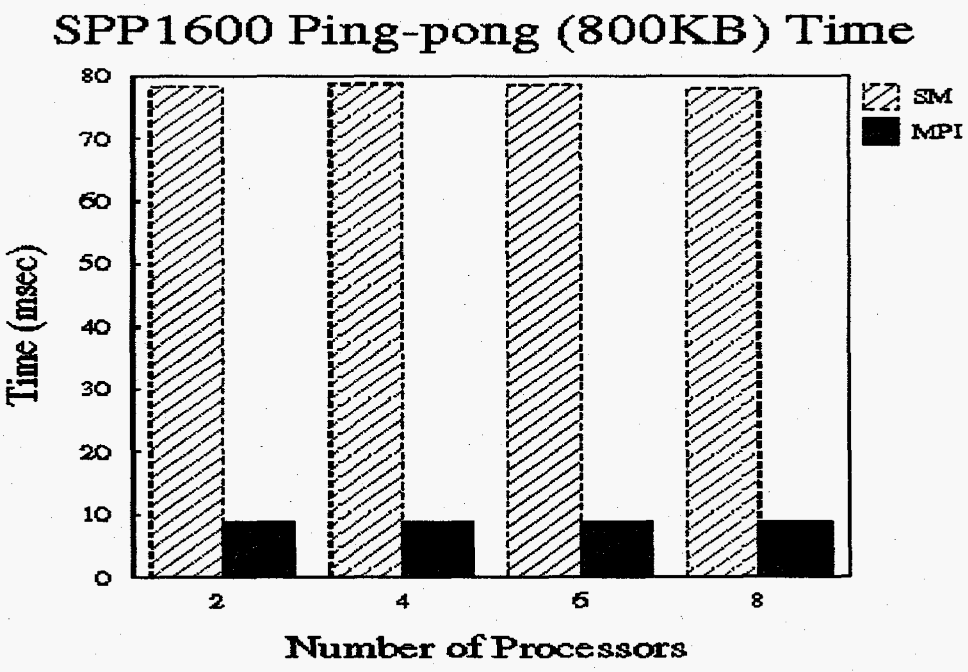

Figure 15. SPP1600 ping-pong Time

The performance of shared memory broadcast and reduction on this SPP1600

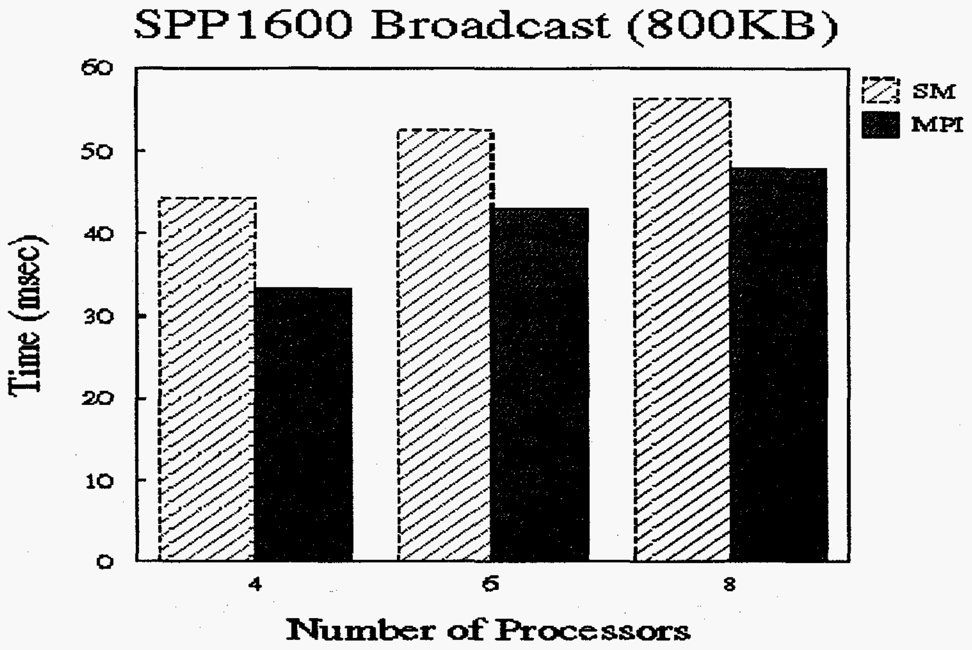

Figure 16. SPP1600 Broadcast Time

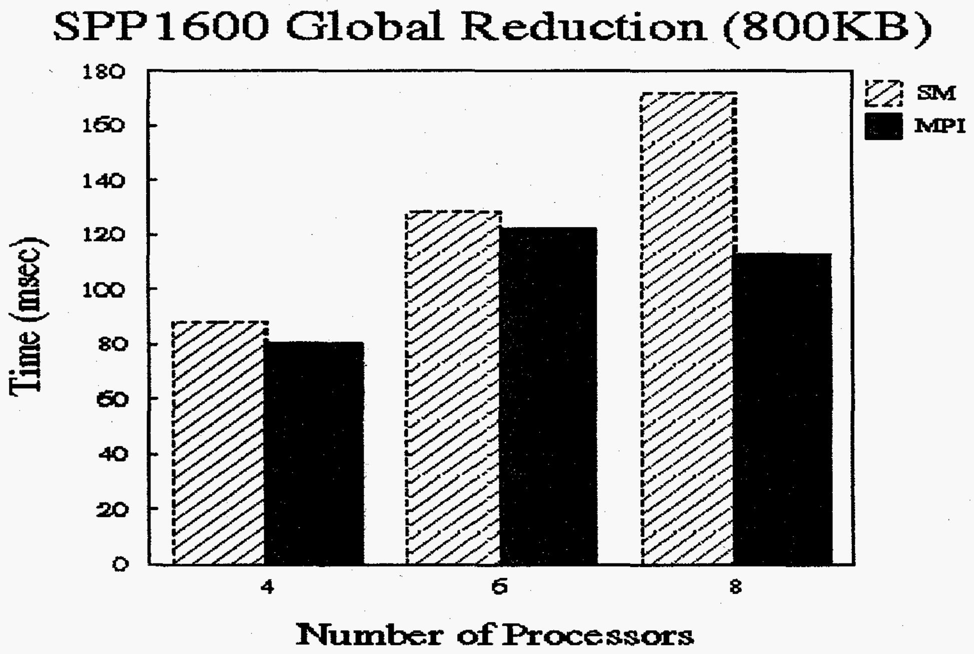

Figure 17. SPP1600 Reduction Time

(Figure 16 and 17) is similar to the other two machines. The queue length for reading the shared block and the cost from the critical section are the major effects in broadcast and reduction respectively.

Since the details of broadcast and reduction implementation in the Convex version of MPI are unclear at this moment, it is anticipated that the MPI broadcast involves regular synchronizations, just like the situation on the DEC AlphaServer. As for reduction operations, the slightly higher cost on six processors is probably because two of the six processors may not be on the same agent (Figure 2). Therefore, the interaction between these two processors has to go through the crossbar switch.

Conclusions

From the COMOPS benchmark results measured on three shared memory machines, the following conclusions can be made.

1. The MPI implementation on the SGI Power Challenge is generally superior to the others, at least for COMOPS operations.

2. In general, the communication performance for COMOPS operations is better in two customized versions of MPI, the Convex MPI and the SG MPI, than in their corresponding shared memory schemes.

3. On the DEC Turbolaser, the communication performance in the shared memory scheme is slightly better than that in the MPI because of the MPI overhead.

It is clear that customizing the MPI implementation based on the specific hardware architecture is a good way to achieve high performance for message passing operations on a shared memory platform. Also, using direct memory copy, instead of going through an intermediate shared space, is critical to the improvement of the communication performance.

Bibliography

CONVEX Computer Corporation. (1994). *Exemplar Architecture.* 2nd Edition, Doc. No. 081-023430-001. Richardson, TX: CONVEX Press. Nov. 1994

Digital Equipment Corp. (1995). *AlphaServer 8000 Series Configuring Guide.* DEC WWW home page http://www.digital.com/info/alphaserver/alphsrv8400/8000.html#spec.

Fenwick., D.M., Foley, D.J., Gist, W.B., VanDoren, S.R., & Wissell, D. (1995). The AlphaServer 8000 Series: High-End Server Platform Development. *Digital Technical Journal,* Vol. 7 No. 1, 43-65.

Gropp, W., Doss, N., & Skjellum, A. (1996). *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard* (On-line Technical Report) Available at http://www.mcs.anl.gov/mpi/mpicharticle/paper.html. Argonne, IL: Mathematics and Computer Science Division, Argonne National Laboratory.

Reed, J. (1996). *Personal Correspondence About SPP1600.* Nov. 1996.

Salo, E. (1996). *Personal Correspondence About SGI MPI.* Nov. 1996.

Silicon Graphics Inc. (1996). Power Challenge Technical Report. SGI WWW home page http://www.sgi.com/Products/hardware/servers, 1995.