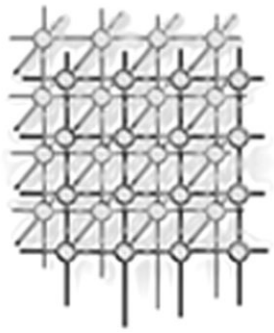
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2003; **15**:779–802 (DOI: 10.1002/cpe.719)

Comparing the performance of MPICH with Cray's MPI and with SGI's MPI



Glenn R. Luecke*,†, Marina Kraeva and Lili Ju

High Performance Computing Group, Iowa State University, Ames, Iowa 50011-2251, U.S.A.

SUMMARY

The purpose of this paper is to compare the performance of MPICH with the vendor Message Passing Interface (MPI) on a Cray T3E-900 and an SGI Origin 3000. Seven basic communication tests which include basic point-to-point and collective MPI communication routines were chosen to represent commonly-used communication patterns. Cray's MPI performed better (and sometimes significantly better) than Mississippi State University's (MSU's) MPICH for small and medium messages. They both performed about the same for large messages, however for three tests MSU's MPICH was about 20% faster than Cray's MPI. SGI's MPI performed and scaled better (and sometimes significantly better) than MPICH for all messages, except for the scatter test where MPICH outperformed SGI's MPI for 1 kbyte messages. The poor scalability of MPICH on the Origin 3000 suggests there may be scalability problems with MPICH. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: MPI; MPICH; performance; Cray T3E-900; SGI Origin 3000

1. INTRODUCTION

The Message Passing Interface (MPI) [1,2] is now commonly used for writing portable message passing programs. MPICH is a public-domain, portable implementation of the MPI Standard that was originally developed and released by Argonne National Laboratory (ANL) and Mississippi State University (MSU) in parallel with the standardization effort. In 1998, MSU implemented a special MPICH (MSU's MPICH) based on Cray's one-sided SHMEM routines for Cray T3E series machines, see [3]. It was unofficially reported that this MPICH gave significantly better performance than Cray's MPI on the Cray T3E-600. Since SHMEM library routines provide a natural, lowest-level interface to remote memory puts and gets, Cray's MPI is also based on SHMEM routines. One of the goals

*Correspondence to: Glenn R. Luecke, High Performance Computing Group, Iowa State University, Ames, Iowa 50011-2251, U.S.A.
†E-mail: grl@iastate.edu

Received 17 October 2001
Revised 1 August 2002

of this paper is to determine the benefit (if any) of using Cray's MPI. Thus, the authors investigate the performance and scalability differences between Cray's MPI and MSU's MPICH on a variety of commonly used communication patterns. The authors did not have access to a Cray T3E-600, so performance comparisons were carried out on the newer Cray T3E-900. In [3], Herbert *et al.* present performance results for MPICH for only ping-pong tests. In this paper, the authors present performance results for seven tests: the ping-pong test with all processors (using mpi_send and mpi_recv), the right-shift test (using mpi_sendrecv), the broadcast test (using mpi_bcast), the reduce test (using mpi_reduce with mpi_sum), the scatter test (using mpi_scatter), the gather test (using mpi_gather) and the all-to-all test (using mpi_alltoall).

A second goal of this paper is to determine the benefit (if any) from using SGI's MPI instead of ANL's MPICH [4]. Thus, the performance and scalability differences between SGI's MPI and ANL's MPICH were investigated. The SHMEM library is available on the SGI Origin machines [5,6], but there is a lower-level interface for remote loads and stores that is also available. SGI has based their MPI on this lower-level interface for both their Origin 2000 and Origin 3000 machines. This performance and scalability comparison is carried out using the same tests described in the above paragraph.

The Cray T3E-900 [7–9] used is a 64-processor machine located in Eagan, Minnesota. Each processor is a DEC Alpha EV5 microprocessor running at 450 MHz with primary instruction and data caches of 8×1024 bytes and one 96×1024 byte secondary cache used for both of them. The communication network is a three-dimensional, bi-directional torus with a bandwidth of 350 Mbytes s^{-1} and a latency of 1.8 μs. The following software was used for the Cray T3E-900: the UNICOS/mk2.0.5 operating system, the Fortran compiler version 3.2.0.1 with the -O3 compiler option, the Cray MPI library version 1.4.0.3, and the above MSU's MPICH library.

The Origin 3000 used is a 512-processor (128 nodes) machine located in Eagan, Minnesota. Each node has four 400 MHz MIPS R12000 processors sharing a common memory and each processor has two levels of cache: 32×1024 byte primary instruction and data caches, and a $8 \times 1024 \times 1024$ byte secondary cache for both data and instructions. The communication network is a hypercube for up to 64 processors and is called a 'fat bristled hypercube' for more than 64 processors since multiple hypercubes are interconnected via a CrayLink Interconnect. The following software was used for the SGI Origin 3000: the IRIX 6.5 operating system, the Fortran compiler version 7.313 with the -O3 -64 compiler options, the SGI MPI library version mpt1.5.3 and the 64-bit MPICH library (version 1.2.2.3) [4]. The MPICH was configured with device ch_shmem and architecture IRIX64. The environment variable PROCESSOR_COUNT was set to 512.

The tests were run with 2, 4, 8, 16, 32, 48 and 64 processors for the Cray T3E-900. As the SGI Origin 3000 has a hypercube communication network we wanted to explore if the time would grow when the next dimension of hypercube was used to run processes. As a result the tests were run with 2, 4, 8, 16, 17, 32, 33, 48, 49, 64, 65, 80, 81, 96, 97, 112, 113, 126 and 128 processors for the Origin. All tests were executed on both machines dedicated to running only these tests. All messages used 8 byte reals. The tests were run with messages of size 8 bytes, 10 kbytes and 1 Mbyte for the ping-pong, the right-shift, the broadcast and the reduce tests. Messages of size 8 bytes, 1 kbytes and 100 kbytes were used for the scatter, the gather and the all-to-all tests because of memory limitations. The tests were run using the default environmental setting for the Cray T3E-900. SGI recommended we set the SMA_GLOBAL_ALLOC, MPI_DSM_MUSTRUN, MPI_BUFFER_MAX = 2000 and MPI_XPMEM_ON (when there are less than 65 processes) environment variables. This was done for all the tests run on the SGI Origin 3000.

The timing methodology is discussed in Section 2. Section 3 presents each test and their performance results. The conclusions are discussed in Section 4.

2. TIMING METHODOLOGY

All tests used the following timing template:

```
integer,parameter :: ntrials=51
real*8, dimension(ntrials) :: array_time,max_time
real*8 :: flush(ncache)
...
do k = 1, ntrials
   flush(1:ncache) = flush(1:ncache) + 0.001
   call mpi_barrier(MPI_comm_world,ierror)
   t = mpi_wtime()

   ... MPI code to be timed ...

   array_time(k) = mpi_wtime() - t
   call mpi_barrier(mpi_comm_world,ierror)
   A(1) = A(1) + flush(mod(k-1,10)+1)
enddo
call mpi_allreduce(array_time,max_time,ntrials,mpi_real8,mpi_max,&
            mpi_comm_world,ierror)
write(*,*)  flush(1)+max_time(1)+A(k)
```

Timings were done by first flushing the cache on all processors by changing the values of the real array `flush(1:ncache)`, prior to timing the desired operation. The value **ncache** was chosen to make the size of the array `flush` twice the size of the secondary cache for the T3E-900 and the Origin 3000, in order to guarantee that the secondary cache is flushed completely. The statement `A(1) = A(1) + flush(mod(k-1,10)+1)`, where `A` is an array involved in the communication, was used to prevent some compilers from removing the cache flushing statement out from the timing loop. The last statement `write(*,*)flush(1)+max_time(1)+A(k)` was used to keep the compilers from considering the code to be timed as 'dead code' and eliminating it. Each processor saves its local time in the array `array_time` and then the maximal time over all processors is stored in the array `max_time` (as was done in [10]).

The resolution of the MPI timer routine `mpi_wtime` is 0.013 μs on the T3E and 0.8 μs on the Origin 2000. This is accurate enough to time the tests in this paper. Each test was performed `ntrials = 51` times when possible. However, on the Origin 3000 some of the MPICH tests (like the all-to-all test) would 'hang' and execution would not complete when using more than 64 processors. Thus for the scatter, the gather and the all-to-all tests the number of trials was reduced from 51 to 15. For all tests on both machines, the first measured timing was usually significantly larger than most of the subsequent times (likely because of start-up overhead), so the first measured time was always thrown away.

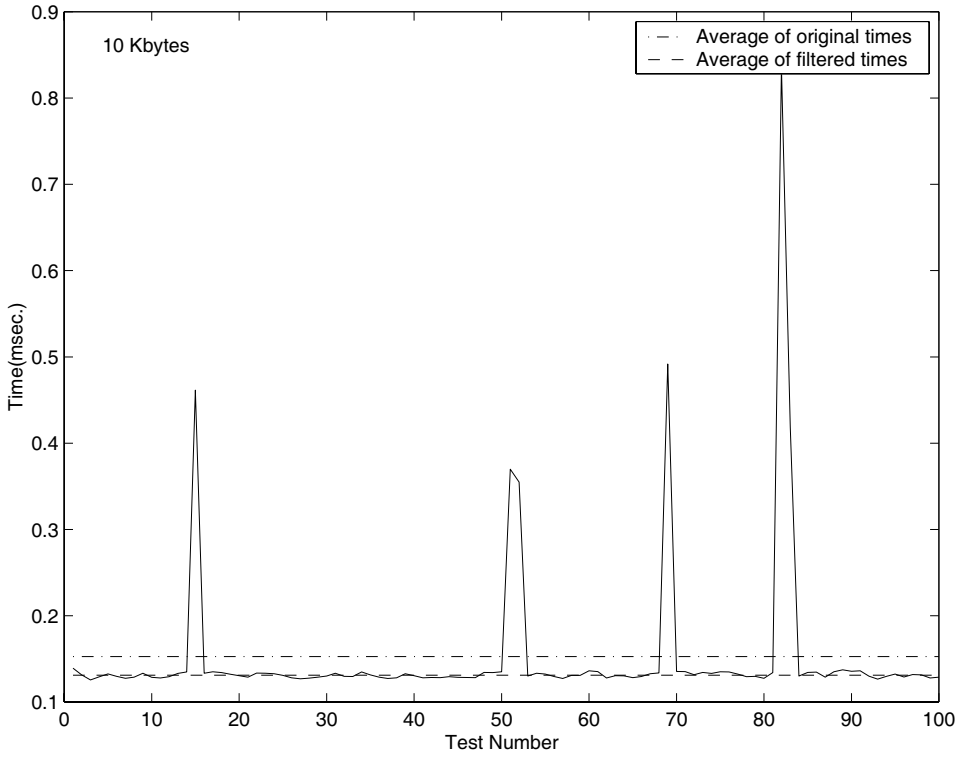


Figure 1. Execution times of Cray's MPI on the T3E-900 for 100 trials of the right-shift test with 64 processors and a 10 kbyte message.

Table I. Standard deviation for the filtered data.

Machine	Average standard deviation (%)	Maximum standard deviation (%)
T3E-900	2.32	18.24
Origin 2000	11.13	55.37

Figure 1 shows that sometimes when repeating the timing of a test, large increases, i.e. 'spikes' in the times occur. These 'spikes' are likely caused by operating system interruptions. This kind of behavior is found in some tests on the T3E-900 and almost in all tests on the Origin 3000. It was considered to be appropriate to remove these 'spikes' by the following filtering process: first the median was computed, then the times greater than 1.8 times the median (however not more than 10% of the data) were removed. Thus up to five largest times out of 51 trials could be removed by the filtering procedure for every test. In Figure 1, the average of the original times is 152.6 μs, but the average of the filtered times is 131.1 μs. This filtering process led to the removal of 0.48% of all the trials for all tests for the T3E-900 and 2.51% of all the trials for the Origin 3000. Table I reports the average and maximum standard deviations for the filtered data over all tests. These results are similar to the results reported in [10].

3. COMMUNICATION TESTS AND PERFORMANCE RESULTS

3.1. The ping-pong with all processors test

The purpose of this test is to determine the performance differences for sending messages between 'close' and 'distant' processors. Round trip times ('ping-pong') between processor 0 and other processors are measured. The round trip times are then divided by two to obtain the average one-way communication time. The code below was used for this test, where $j = 1$ to 63 for the T3E-900 and $j = 1$ to 127 for the Origin 3000:

```
real*8 :: A(n),B(n)
   . . .
   if (myrank == 0) then
      call mpi_send(A,n,mpi_real8,j,1,mpi_comm_world,ierror)
      call mpi_recv(B,n,mpi_real8,j,2,mpi_comm_world,status,ierror)
   else
      if ((myrank == j) then
         call mpi_recv(A,n,mpi_real8,0,1,mpi_comm_world,status,ierror)
         call mpi_send(B,n,mpi_real8,0,2,mpi_comm_world,ierror)
      endif
   endif
```

Notice that the rank 0 processor sends `A` and the rank j processor receives `A` and then sends `B` to the rank 0 processor. If the rank j processor sent `A` instead of `B`, then `A` would likely be in the cache and the return trip would be faster.

Figure 2 presents the performance data for this test on the T3E-900. Cray's MPI on the T3E-900 showed excellent scalability for all three message sizes and MSU's MPICH scaled nearly as well. Cray's MPI outperformed MSU's MPICH for messages of size 8 bytes by a factor of three, and by a factor of two for messages of size 10 kbyte. They performed about the same for 1 Mbyte messages.

Figure 3 presents the performance data for this test on the Origin 3000. SGI's MPI performed at least three times better than MPICH for all messages for this test.

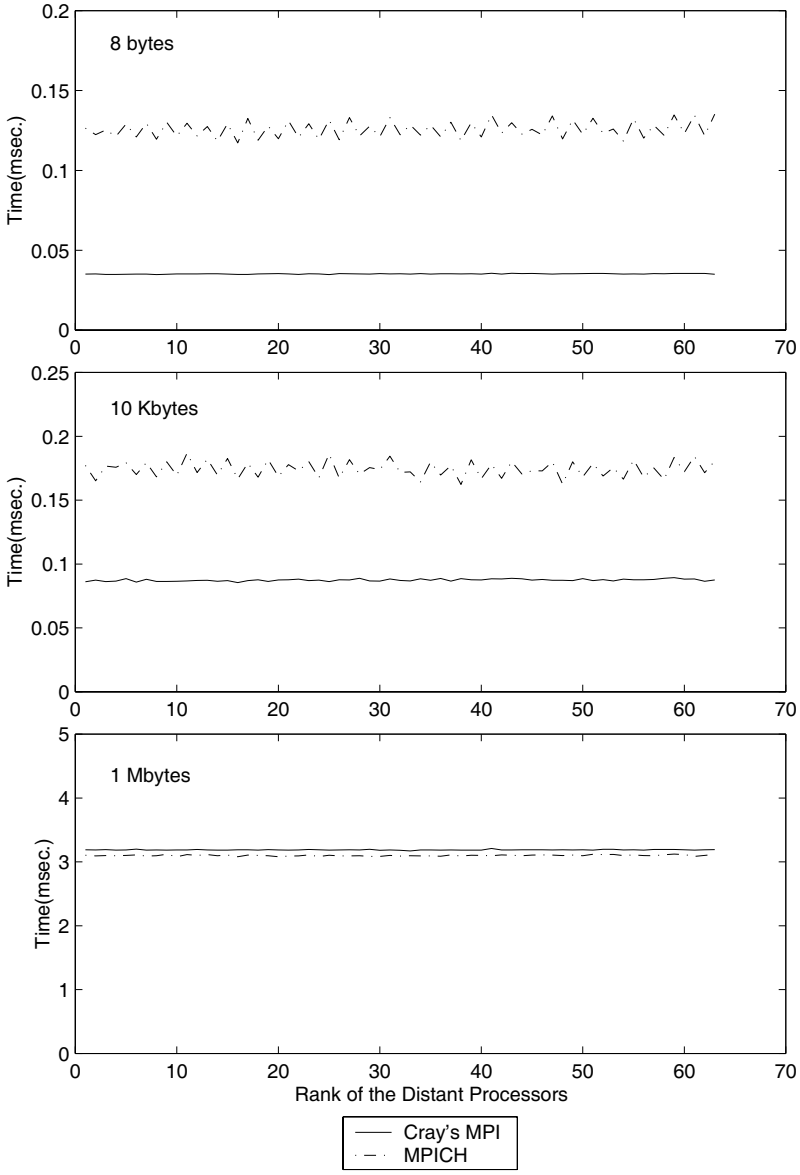


Figure 2. The ping-pong with all processors test for the T3E-900.

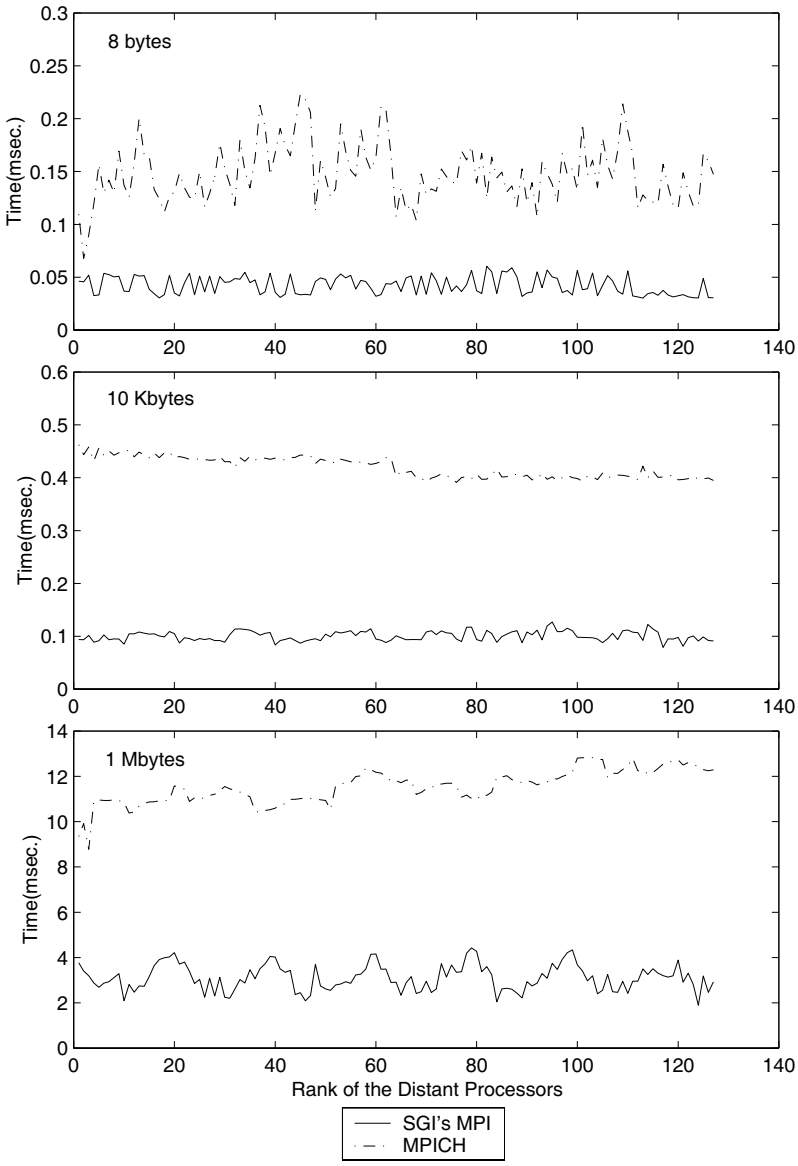


Figure 3. The ping-pong with all processors test for the Origin 3000.

3.2. The circular right-shift test

The purpose of this test is to measure the time to perform a 'circular right-shift', i.e. each processor sends array `A` to its 'right' neighbor and receives the message into array `B` from its 'left' neighbor. The code used for this test was:

```
real*8 :: A(n),B(n)

call mpi_sendrecv(A,n,mpi_real8,modulo(myrank+1,p),100,B,n,mpi_real8,&
            modulo(myrank-1,p),100,mpi_comm_world,status,ierror)
```

Observe that in an 'ideal' parallel computer, the operations for the circular right-shift may be done concurrently and hence the time for this operation would be independent of the number of processors used in communications. Figure 4 shows this is nearly the case for the Cray T3E-900 using both Cray's MPI and MSU's MPICH.

Figure 4 presents the performance data for this test on the T3E-900. Cray's MPI outperformed MSU's MPICH for messages of size 8 bytes by a factor of four, and messages of size 10 kbytes by a factor of 2.5. They performed nearly the same for 1 Mbyte messages.

Figure 5 presents the performance data for this test on the Origin 3000. SGI's MPI performed better than MPICH for all messages. MPICH scaled almost as well as SGI's MPI for the 8 byte messages. With the growth of the message size SGI's MPI demonstrated significantly better scalability than MPICH. MPICH scaled poorly for the medium and large messages. To better evaluate the scalability of SGI's MPI for this test, Figure 6 shows the data for SGI's MPI only. Notice that even though SGI's MPI scaled well compared with MPICH (see Figure 5), SGI's MPI does not scale nearly as well as both MPIs on the Cray T3E-900.

In Figure 5 one can see several 'spikes' both for SGI's MPI and MPICH. When these tests were rerun there were still 'spikes' but they occurred at different numbers of processes. To find out what was causing this, the authors examined the raw data when the 'spikes' occurred. Figure 7 illustrates this. Notice that there are about 17 out of the 51 timings that are 'reasonable'. Even though we ran all tests with no other user jobs executing, there was some activity occurring (probably due to operating system activity).

3.3. The broadcast test

The purpose of this test is to measure the time of broadcasting an array `A` from one processor to the other $p - 1$ processors. The code used for this test was:

```
real*8 :: A(n)

call mpi_bcast(A,n,mpi_real8,0,mpi_comm_world,ierror)
```

Figure 8 presents the performance data for this test on the T3E-900. Both Cray's MPI and MSU's MPICH showed good scalability. Cray's MPI outperformed MSU's MPICH for the 8 byte message by a factor of two, and for the 10 kbyte message by a factor of 1.5. However, MSU's MPICH performed a little better than Cray's MPI for the 1 Mbyte message.

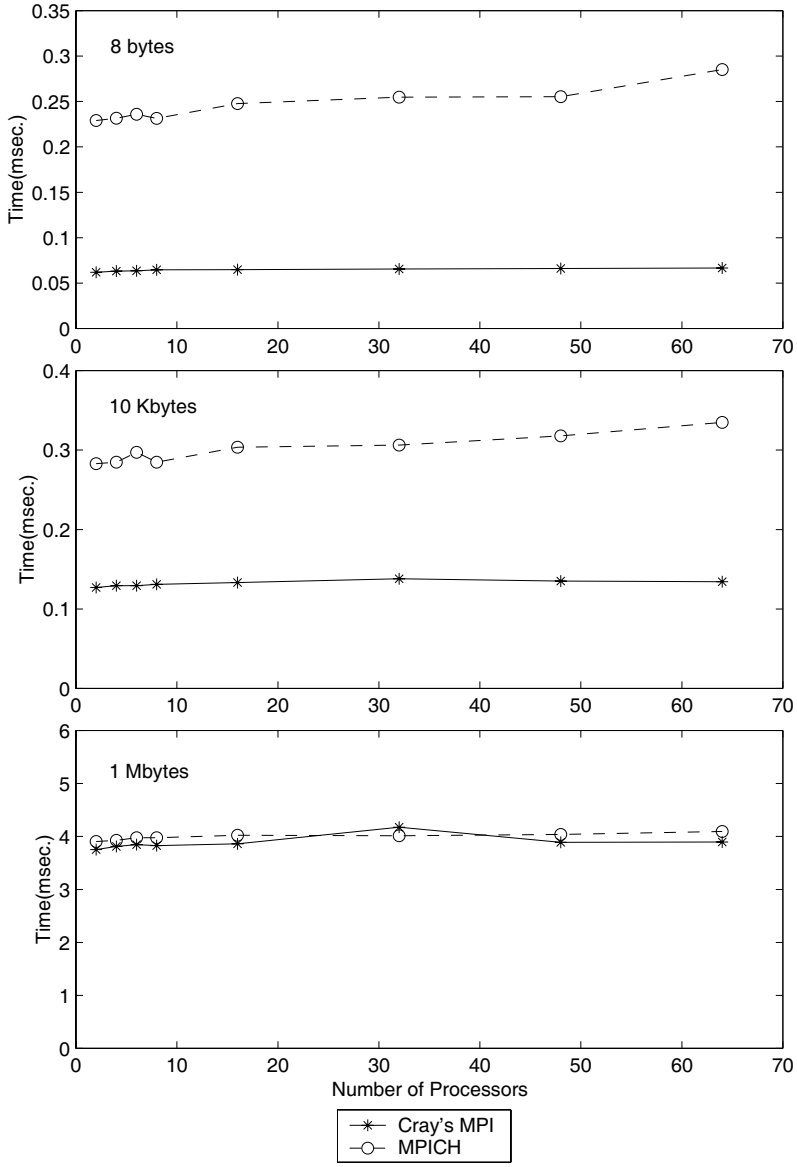


Figure 4. The circular right-shift test for the T3E-900.

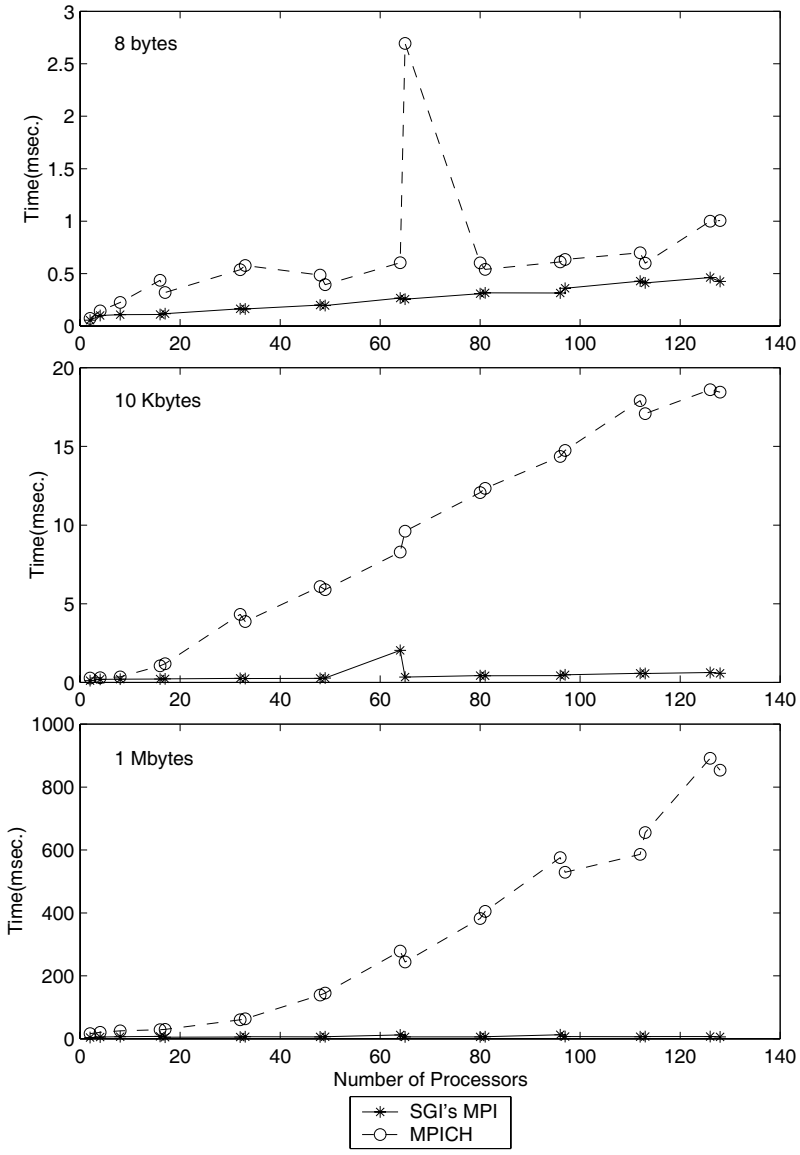


Figure 5. The circular right-shift test for the Origin 3000.

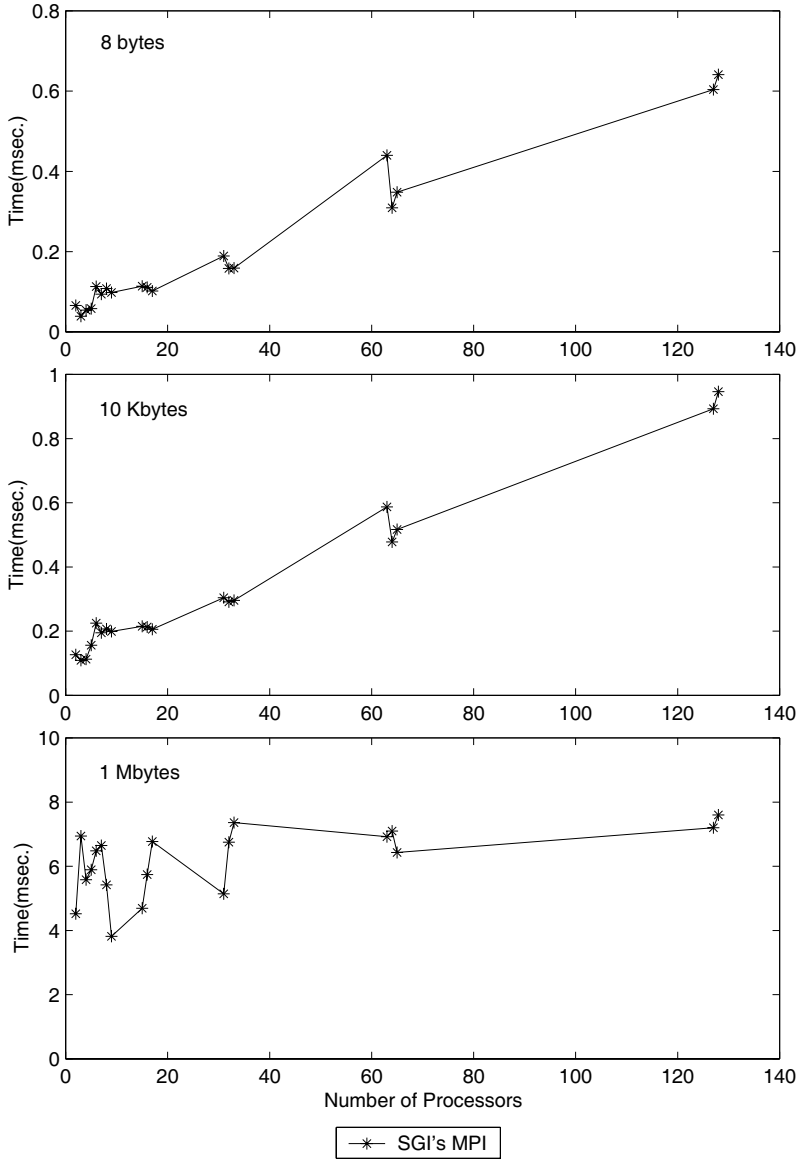


Figure 6. The circular right-shift test for the Origin 3000.

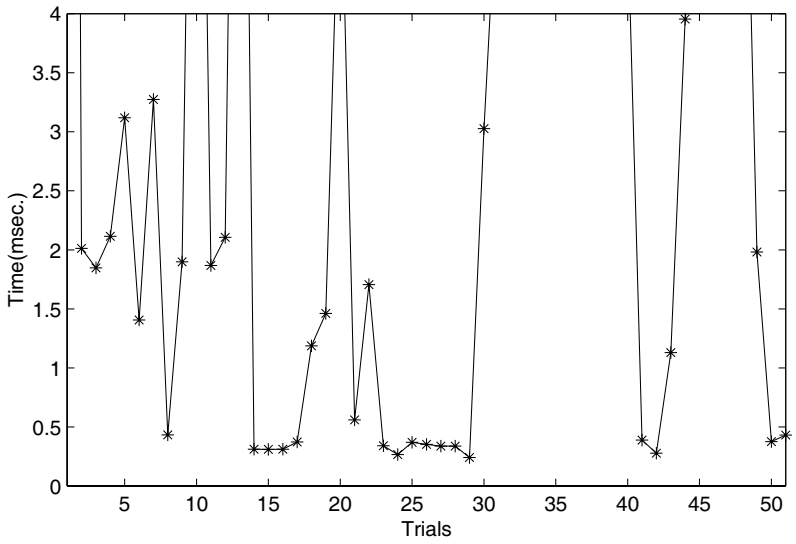


Figure 7. Execution times of MPICH on the Origin 3000 for 51 trials of the circular right-shift test with 65 processors and an 8 byte message.

Figure 9 presents the performance data for this test on the Origin 3000. Similar to the circular right-shift test, SGI's MPI performed and scaled much better than MPICH for medium and large messages. They performed almost the same for the 8 byte message.

3.4. The reduce test

The purpose of this test is to measure the time required to add the `A` arrays from each processor and place this sum in array `B` on processor 0. Unlike the other communication tests in this paper this test involves floating point operations. The code used for this test was:

```
real*8 :: A(n),B(n)

call mpi_reduce(A,B,n,mpi_real8,mpi_sum,0,mpi_comm_world,ierror)
```

Figure 10 presents the performance data for this test on the T3E-900. Cray's MPI outperformed MSU's MPICH for all messages in this test while both showed good scalability.

Figure 11 presents the performance data for this test on the Origin 3000. SGI's MPI significantly outperformed MPICH in this test. The results of this test are similar to those of the broadcast test on the Origin 3000.

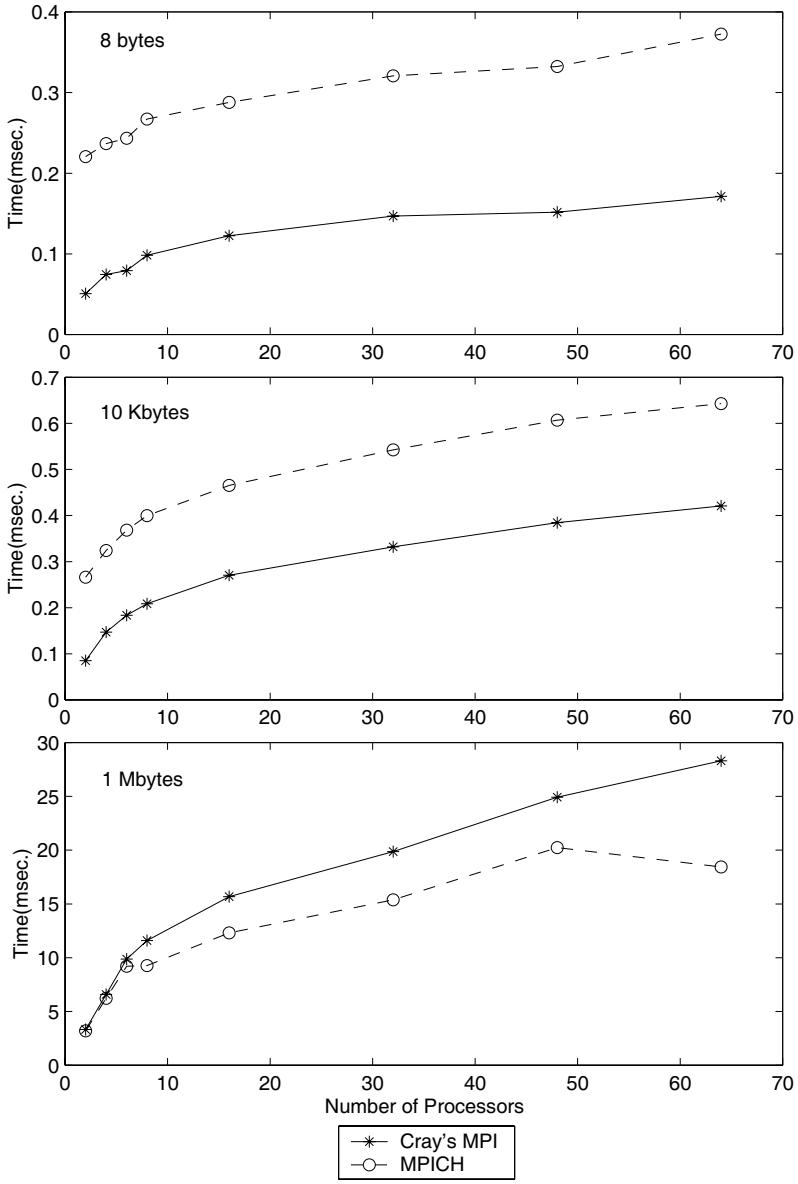


Figure 8. The broadcast test for the T3E-900.

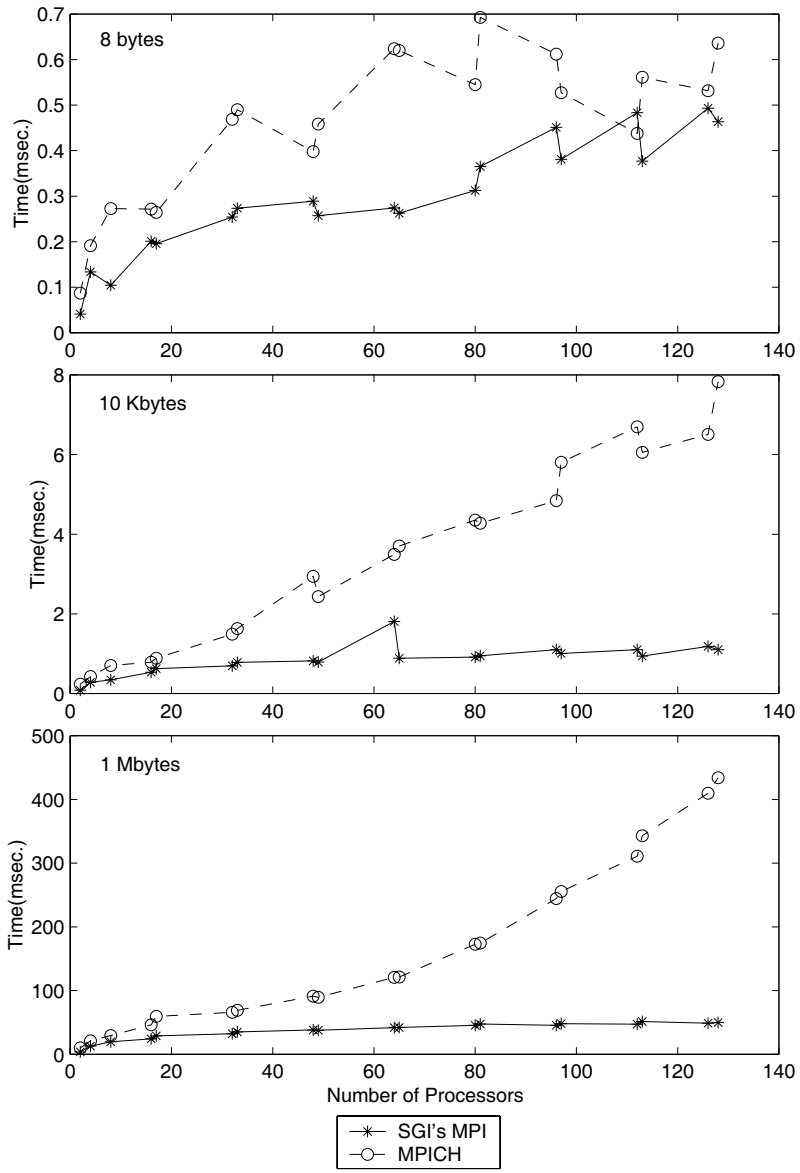


Figure 9. The broadcast test for the Origin 3000.

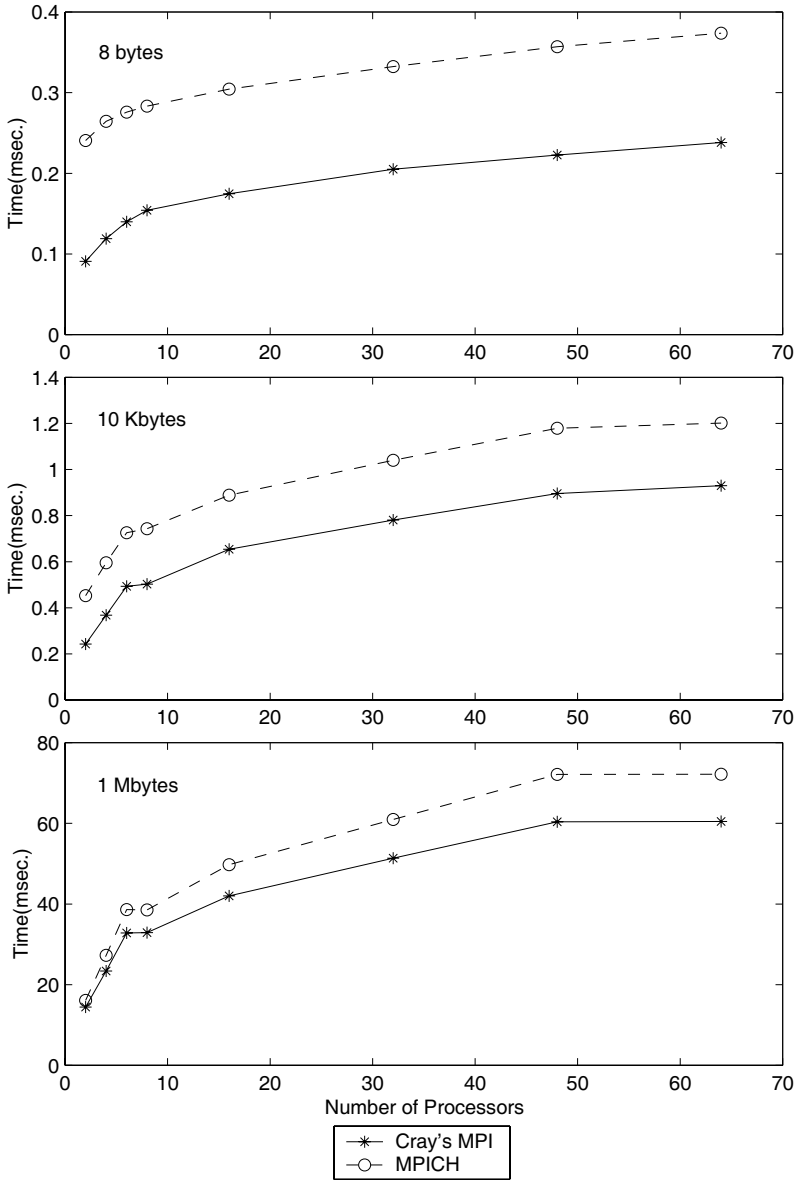


Figure 10. The reduce test for the T3E-900.

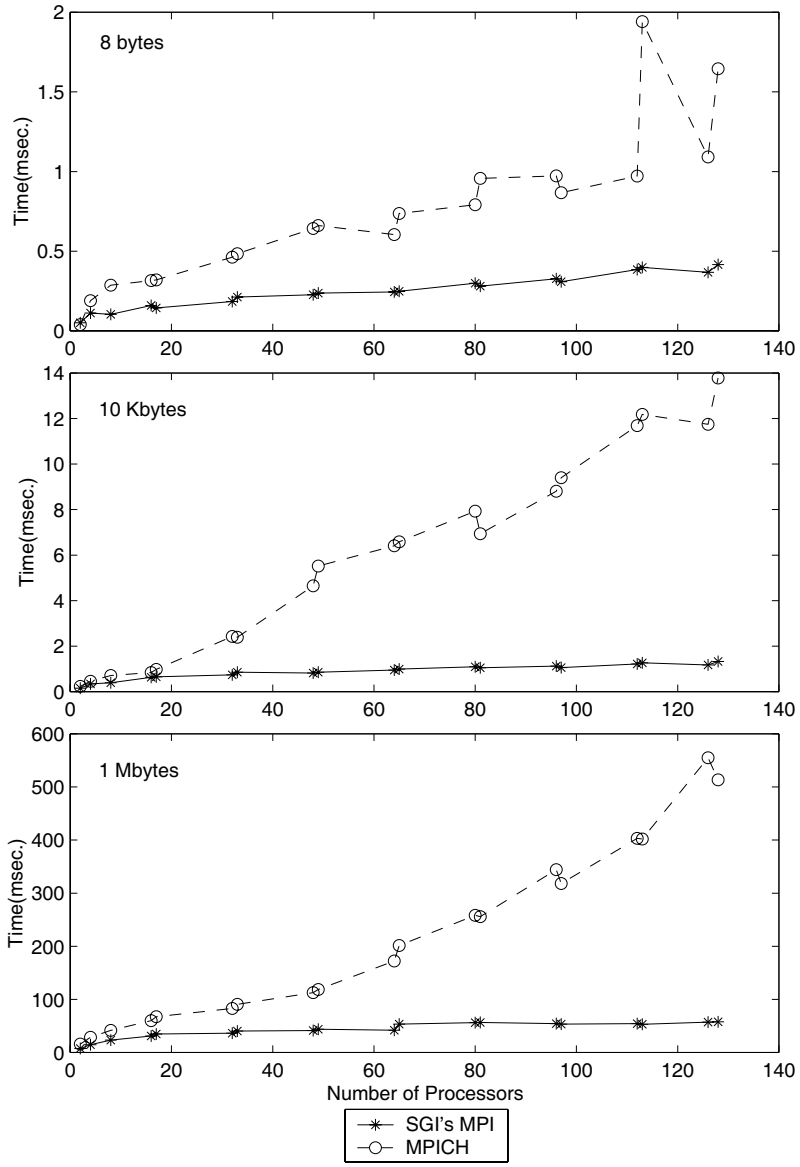


Figure 11. The reduce test for the Origin 3000.

3.5. The scatter test

The purpose of this test is to measure the time to scatter the data in array `A` on processor 0 to the `B` arrays on each processor. The code used for this test was:

```
real*8 :: A(n*p),B(n)

call mpi_scatter(A,n,mpi_real8,B,n,mpi_real8,0,mpi_comm_world,ierror)
```

Figure 12 presents the performance data for this test on the T3E-900. Cray's MPI outperformed MSU's MPICH for messages of sizes 8 bytes and 1 kbyte, but MSU's MPICH performed slightly better than Cray's MPI for the message of size 100 kbytes.

Figure 13 presents the performance data for this test on the Origin 3000. Using MPICH this test would 'hang' and execution would not complete when running more than 100 processes. However, when the number of trials was reduced from 51 to 15, the test completed successfully, so performance results are based on 15 trials. Similar to the previous tests SGI's MPI performed a little better than MPICH for the message of size 8 bytes, and much better for the large message due to the scalability problems of MPICH. However, MPICH outperformed SGI's MPI for 1 kbyte messages. The result is repeatable. This is the only one out of seven tests where MPICH performed better than SGI's MPI.

3.6. The gather test

The gather test is the inverse operation to the scatter test. The code used for this test was:

```
real*8 :: A(n),B(n*p)

call mpi_gather(A,n,mpi_real8,B,n,mpi_real8,0,mpi_comm_world,ierror)
```

Figure 14 presents the performance data for this test on the T3E-900. Cray's MPI outperformed MSU's MPICH for messages of sizes 8 bytes and 10 kbytes, but for the message of size 100 kbytes MSU's MPICH performed a little better than Cray's MPI.

Figure 15 presents the performance data for this test on the Origin 3000. SGI's MPI significantly outperformed MPICH for all messages for this test. Since the gather operation is the inverse of the scatter operation, one might expect their performance results to be similar. This is true for the Cray T3E-900. However, for the Origin 3000 the performance and scalability of MPICH for the gather test is significantly worse than that of the scatter test.

3.7. The all-to-all test

The purpose of this test is to measure the time needed for each processor to send distinct data to the other $p - 1$ processors and to receive data from the other processors. Notice the heavy communication traffic generated by this test.

```
real*8 :: A(n*p),B(n*p)

call mpi_alltoall(A,n,mpi_real8,B,n,mpi_real8,mpi_comm_world,ierror)
```

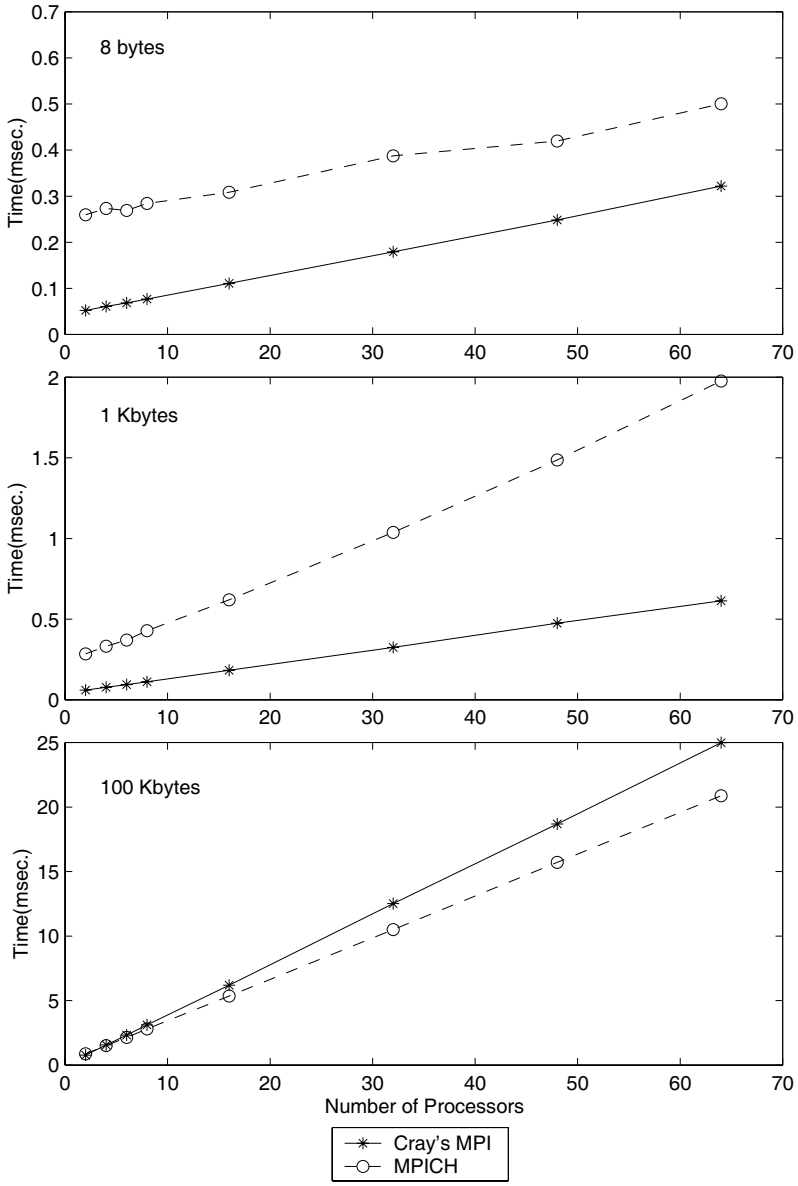


Figure 12. The scatter test for the T3E-900.

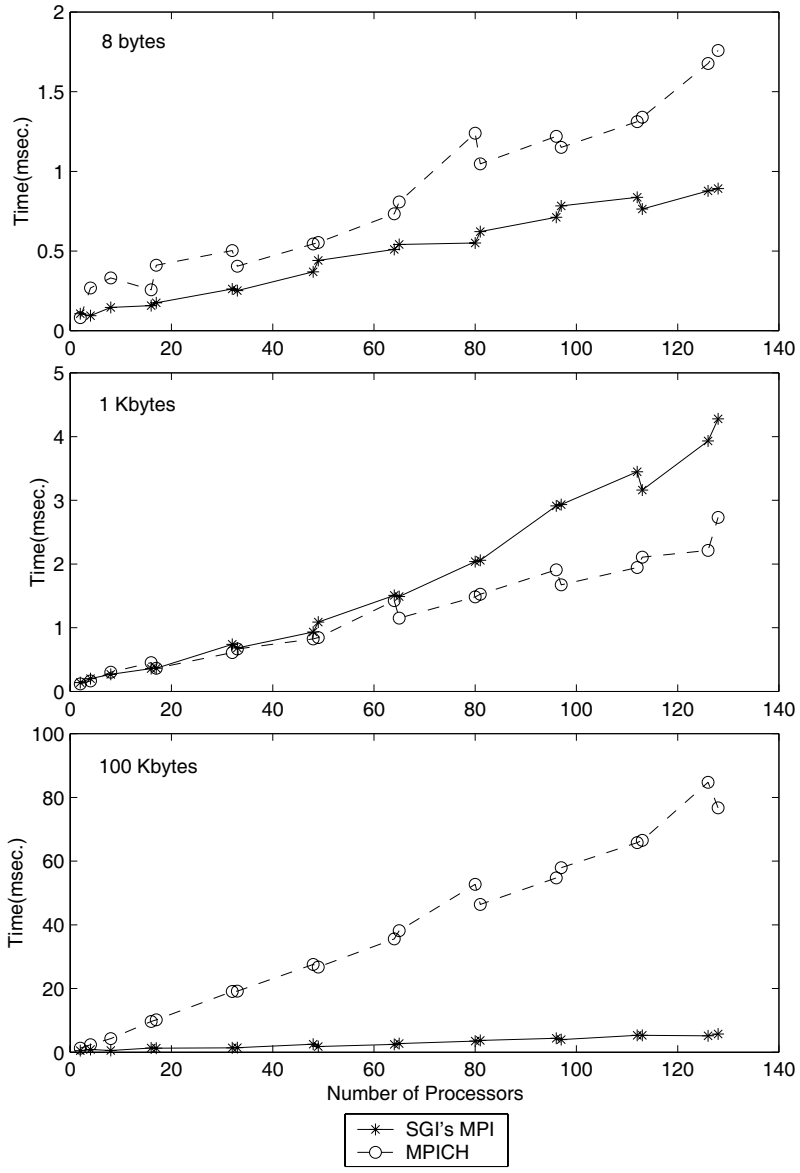


Figure 13. The scatter test for the Origin 3000.

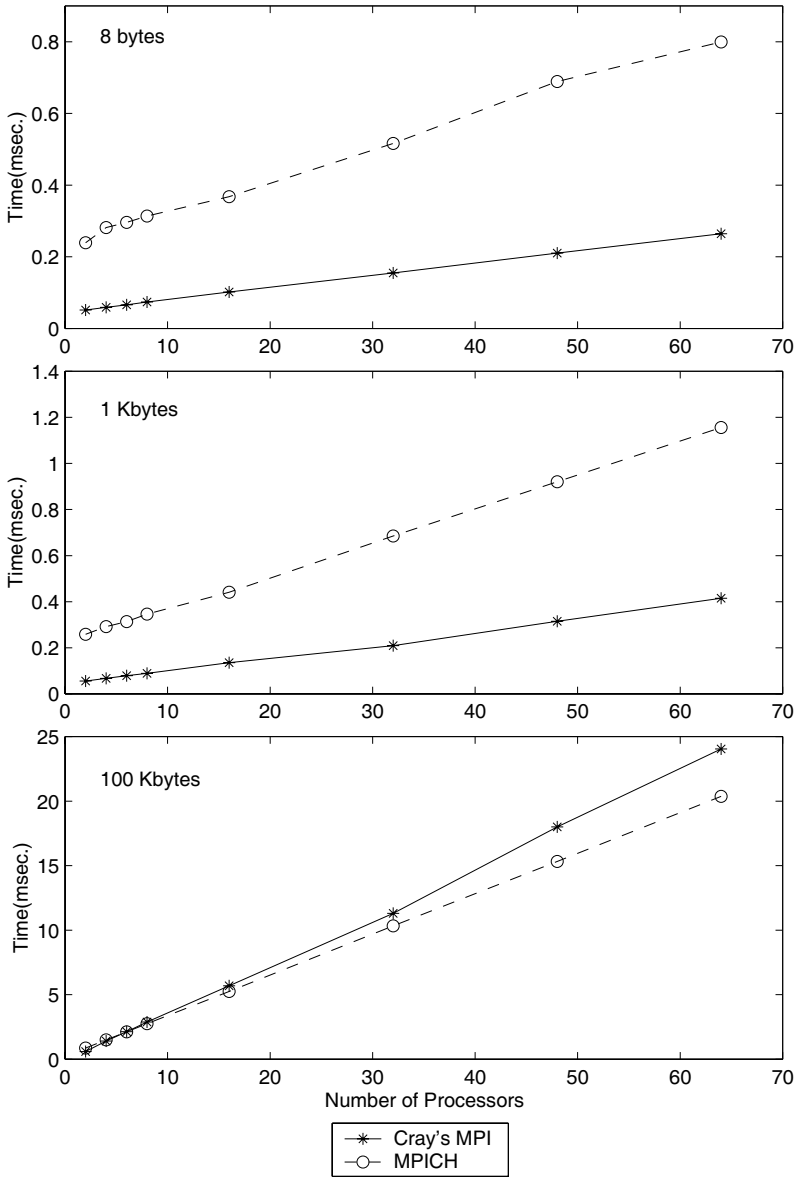


Figure 14. The gather test for the T3E-900.

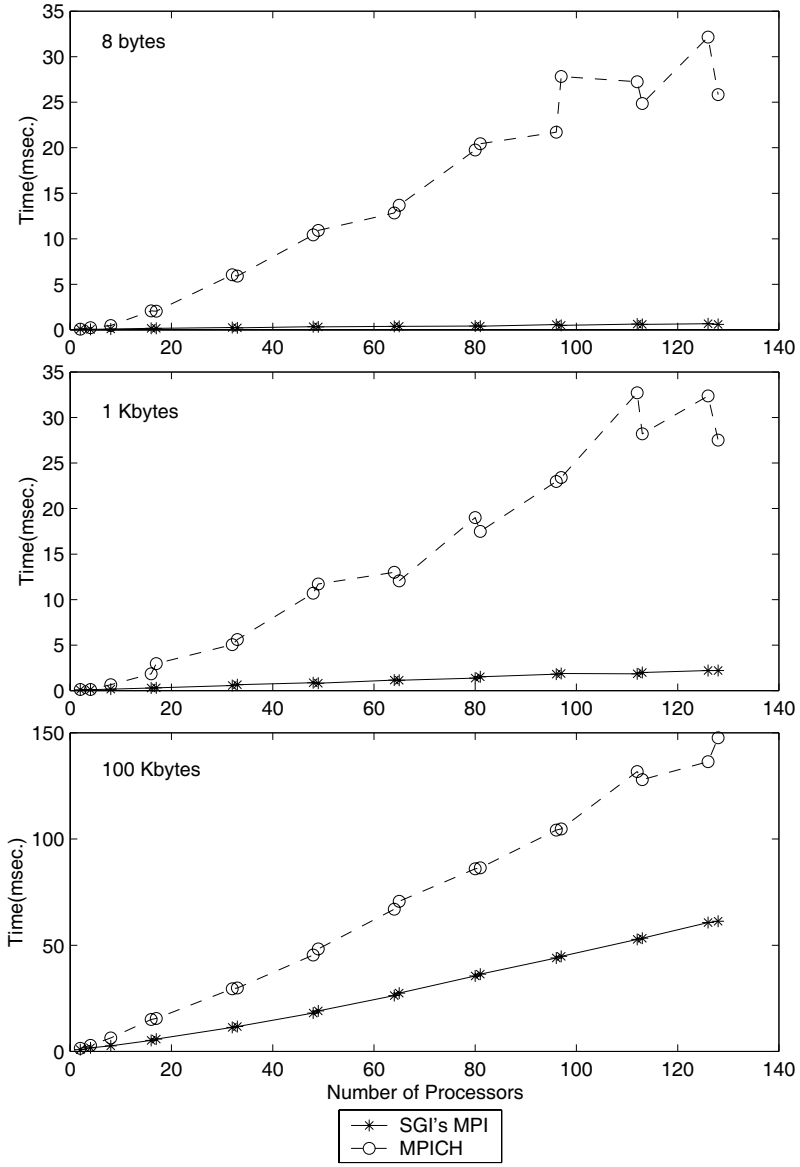


Figure 15. The gather test for the Origin 3000.

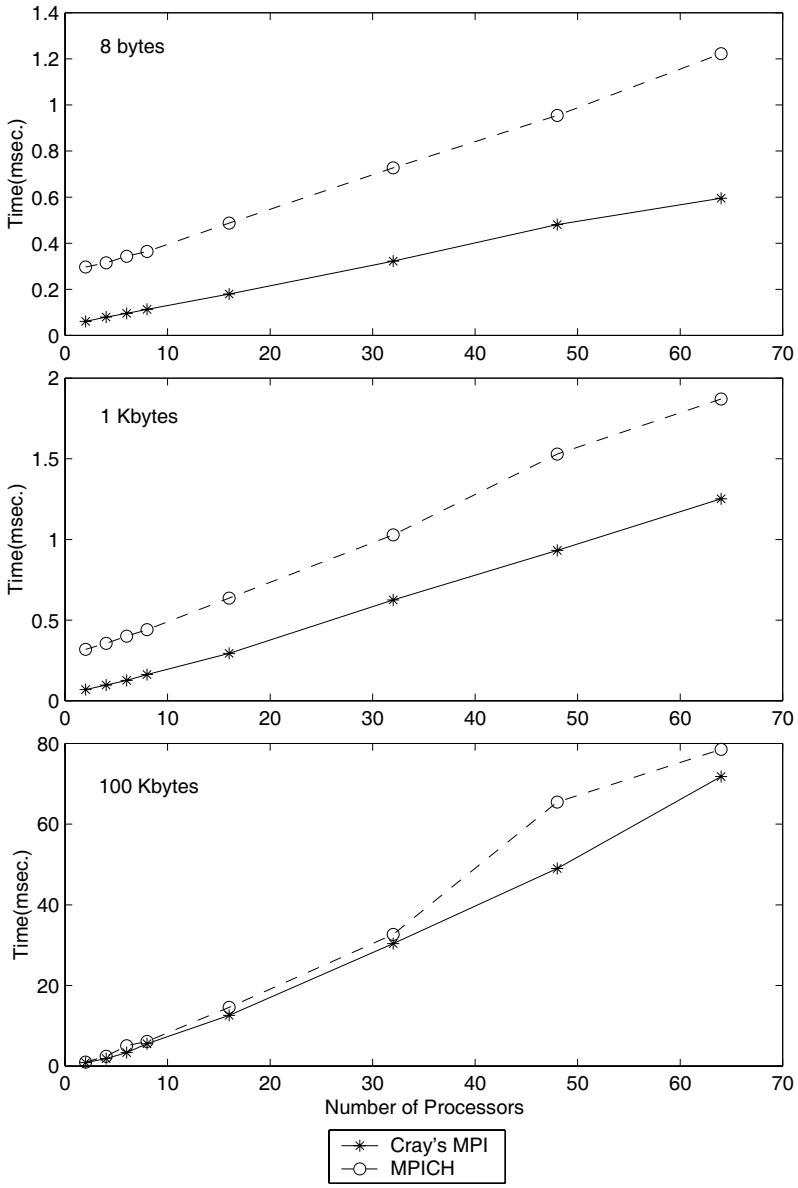


Figure 16. The all-to-all test for the T3E-900.

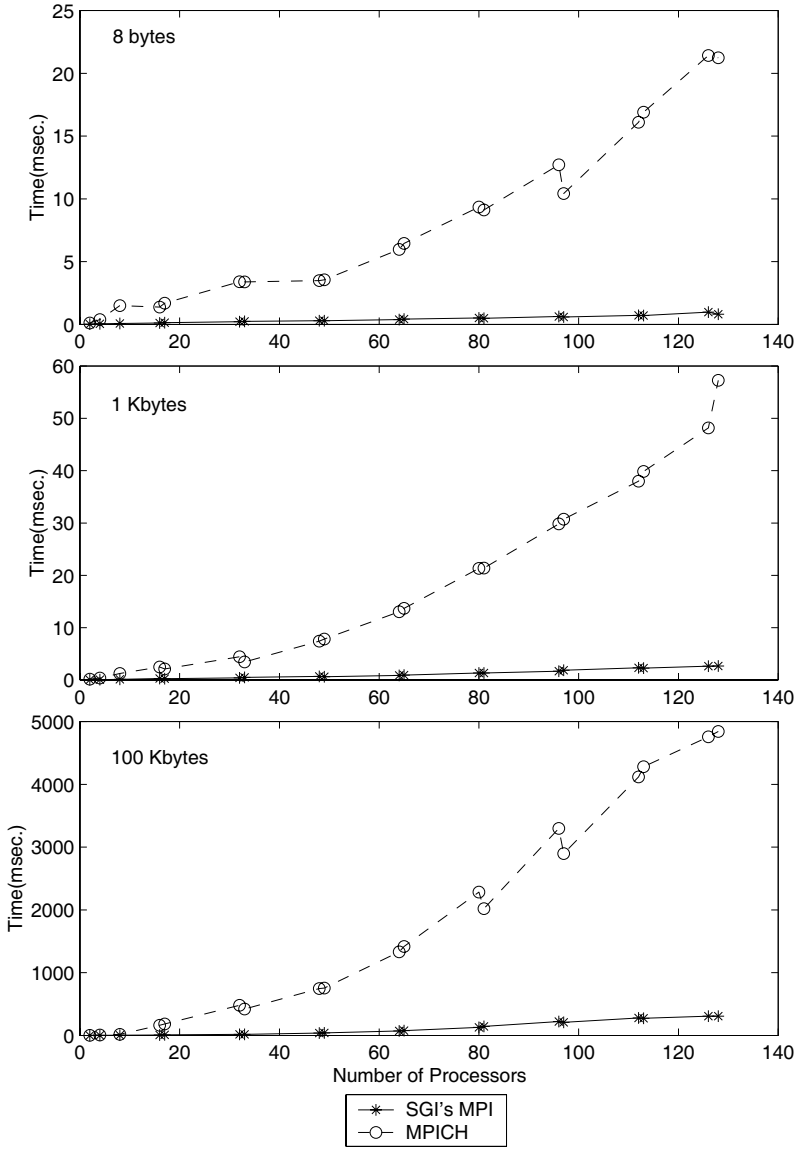


Figure 17. The all-to-all test for the Origin 3000.

Figure 16 presents the performance data for this test on the Cray T3E-900. Cray's MPI outperformed MSU's MPICH for messages of sizes 8 bytes and 1 kbyte, and they performed about the same for messages of size 100 kbytes. Figure 17 presents the performance data for this test on the Origin 3000. SGI's MPI performed much better than MPICH for all messages for this test. Similar to the gather test, MPICH has a scalability problem for all the messages tested.

4. CONCLUSIONS

This paper compares the performance of MPICH and the vendor MPI on a Cray T3E-900 and a SGI Origin 3000 for seven commonly-used communication tests with messages of different sizes. Cray's MPI performed better (and sometimes significantly better) than MSU's MPICH for small and medium messages. They both performed about the same for the large messages for all tests except for the broadcast, scatter and gather tests where MSU's MPICH was about 20% faster. Both Cray's MPI and MSU's MPICH scaled about the same.

SGI's MPI performed and scaled better (and sometimes significantly better) than MPICH for all messages, except for the scatter test where MPICH outperformed SGI's MPI for 1 kbyte messages. It was found that MPICH often did not scale well on the Origin 3000 when compared with the SGI's MPI. These results suggest there may be scalability problems with MPICH.

Both Cray's and SGI's MPI generally provide significant performance improvements over MPICH for small and medium sized messages making it worthwhile (from the user's point of view) that Cray and SGI provide optimized versions of MPI. Those tests where MPICH or MSU's MPICH outperformed vendor MPI demonstrate that both Cray and SGI can make further improvements to their MPI.

ACKNOWLEDGEMENTS

We would like to thank Cray Inc. and SGI for allowing us to use their Cray T3E-900 and Origin 3000 located in Eagan, Minnesota.

REFERENCES

1. MPI Standard Web Site. http://www-unix.mcs.anl.gov/mpi/index.html.
2. Snir M, Otto SW, Huss-Lederman S, Walker DW, Dongarra J. *MPI, The Complete Reference (Scientific and Engineering Computation)*. MIT Press, 1996.
3. Hebert LS, Seefeld WG, Skjellum A. MPICH on the Cray T3E. *Report CEWES MSRC/PET TR/98-31*, 1998. http://www.erc.msstate.edu/labs/hpcl.
4. MPICH home page. http://www-unix.mcs.anl.gov/mpi/mpich/.
5. Silicon Graphics. Origin server. *Technical Report*, April 1997.
6. Fier J. *Performance Tuning Optimization for Origin 2000 and Onyx 2*. Silicon Graphics, 1996. http://techpubs.sgi.com.
7. Cray Web Server. http://www.cray.com.
8. Cray Inc. *CRAY T3E Programming with Coherent Memory Streams*, December 18, 1996.
9. Anderson E, Brooks J, Grassl C, Scott S. Performance of the CRAY T3E multiprocessor. *Proceedings ACM/IEE SC97 Conference*, San Jose, CA, 15–21 November 1997.
10. Luecke GR, Raffin B, Coyle JJ. Comparing the communication performance and scalability of a Linux and an NT Cluster of PCs, a SGI Origin 2000, an IBM SP and a Cray T3E-600. *Journal of Performance Evaluation and Modelling for Computer Systems*. http://hpc-journals.ecs.soton.ac.uk/PEMCS/ [2000].

