Comparing the Scalability of the Cray T3E-600 and the Cray Origin 2000 Using SHMEM Routines

Glenn R. Luecke, Bruno Raffin and James J. Coyle
grl@iastate.edu, raffin@iastate.edu, jjc@iastate.edu
Iowa Sate University
Ames, Iowa 50011-2251 USA

Janurary 1999

Abstract

This paper presents relative scalability results for the Cray T3E-600 and the Cray Origin 2000 on five communication tests for a variety of message sizes and for 4, 8, 12, ..., 128 processors. The five communication tests where chosen to represent commonly-used communication patterns with low contention (accessing distant messages, a right shift and a binary tree broadcast) to communication patterns with high contention (a "basic" broadcast and an all-to-all). Both machines scaled roughly the same on all tests except the right shift with short messages where the T3E scaled significantly better than the Origin. However, it should be noted that the T3E outperformed the Origin 2000 for most of these tests.

Keywords: Cray T3E-600; Cray Origin 2000; Relative scalability; Performance evaluation using SHMEM library.

1 Introduction

The ability of a parallel computer to efficiently use large numbers of processors is an important performance criterion. Such a computer is often called "scalable". This paper presents relative scalability results for the Cray T3E-600 and the Cray Origin 2000 on five communication tests for a variety of message sizes and for 4, 8, 12, ..., 128 processors. Limited access to these machines prevented the running of these tests for 1, 2, 3, ..., 128 processors. It should be pointed out that one can purchase a T3E with over a thousand processors whereas the maximum number of processors that can be currently purchased on an Origin is only 128. Thus, the scalability of these machines can only be compared with at most 128 processors.

For the purposes of this paper, **relative scalability** is defined as follows. For a given communication test and for a given message size, plot the graph of the performance ratios, (T3E time)/(Origin time), as a function of the number of processors. If this graph is a horizontal line, then both machines are said to scale the same. If the graph is a line with positive (negative) slope, then the Origin (T3E) is said to scale better than the T3E (Origin). To illustrate this, suppose for a given test and given message size, (T3E Time) = p^2 microseconds and (Origin Time) = p microseconds, where p is the number of processors. Then the graph is a line with slope 1 and clearly the Origin scales better than the T3E. However, for the tests considered in this paper, these graphs are never linear. Since we are primarily interested in the relative behavior of these machines for large numbers of processors, if a graph is "roughly a horizontal line for large numbers of processors", then the two machines are said to scale about the same. If a graph shows "an increasing (decreasing) trend for large numbers of processors", then the Origin (T3E) is said to scale better than the T3E (Origin).

The five communication tests were chosen to represent commonly-used communication patterns with low contention (accessing distant messages, a right shift and a binary tree broadcast) to communication patterns with high contention (a "basic" broadcast and an all-to-all). All tests were written in Fortran 90 using the `shmem_get` and `shmem_put` communication routines [6]. These SHMEM routines were used instead of MPI [13, 10, 9] routines since the SHMEM routines provide the best communication performance on these machines. Moreover this guarantees that the same algorithm is being used for each test on each machine. For example, if `mpi_bcast` had been used to evaluate the performance of the broadcast operation, one machine may use a binary tree algorithm for all message sizes to implement `mpi_bcast` whereas the other machine may use different algorithms depending on the message size [5]. This would make it difficult to compare the scalability of these two machines.

The Origin 2000 used was a 128 processor machine located in Eagan, Minnesota. Each processor is a MIPS R10000, 195 MHz microprocessor with a peak theoretical performance of 390 Mflop/s. Each node consists of two processors sharing a common memory. There are two levels of cache: 32 Kbytes first level instruction and data caches, and a 4 Mbytes second level cache for both data and instructions. The communication network is a hypercube for up to 32 processors and is called a "fat bristled hypercube" for more than 32 processors since multiple hypercubes are interconnected via a CrayLink Interconnect. For more information see [1, 2, 7]. With the `shmem_get8` routine, a maximum node-to-node bandwidth of about 135 Mbytes per second was measured when transmitting a 10 Mbyte message. The MIPSpro version 7.2.1.1m Fortran compiler with the -03 -64 compiler options and sma library version 3.1.1.0 were used.

The T3E used was a 512 processor machine also located in Eagan, Minnesota. Each processor is a DEC Alpha EV5 microprocessor running at 300 MHz, with a peak theoretical performance of 600 Mflop/s. There are two levels of cache: 8 Kbytes first level instruction and data caches and a 96 Kbytes second level cache for both data and instructions. The communication network is a three-dimensional, bi-directional torus. For more information see [1, 12, 11, 3]. With the `shmem_get8` routine, a maximum node-to-node bandwidth of about 336 Mbytes per second was measured when transmitting a 10 Mbyte message. The version 3.1.0.0.2 Fortran compiler with the -03 compiler options and sma library version 3.1.1.0 were used. For both machines, the optimization level used on the compiler did not affect the performance results.

All tests were executed on machines dedicated to running only these tests. All messages used 8 bytes reals. The following message sizes were employed: 8 bytes, 104 bytes, 1 Kbyte, 10 Kbytes, 100 Kbytes, 1 Mbyte and 10 Mbytes, except there was not enough memory to execute the all-to-all test using 1 Mbyte and 10 Mbytes (here 1 Kbyte means 10^3 bytes and 1 Mbyte means 10^6 bytes). The results regarding relative scalability were quite similar between 8 and 104 byte messages. Among the 1 Kbyte to 10 Mbyte messages the relative scalability was also quite similar. For brevity, we present results only for 8 bytes and 1 Mbyte (or 100 Kbytes for the all-to-all test). All tests could have been run with a job requesting 128 processors even though a test might only require 2 processors. When this same test was run requesting only 2 processors instead of 128, the measured performance would usually increase. Thus, tests requiring p processors were run with exactly p processors requested.

Section 2 introduces the timing methodology employed and section 3 presents each of the five tests and performance results. The conclusions are then discussed in section 4.

2 Timing Methodology

Timings were done by first flushing the cache on all processors by changing the values in the real array `cache(1:ncache)`, prior to timing the desired operation. The value `ncache` was chosen so the size of `cache` was the size of the secondary cache, 4 Mbytes for the Origin and 96 Kbytes for the T3E.

```
parameter (ntest=51)
real*8, dimension(ntest) ::  time, array_time
.  .  .
do k = 1, ntest
   cache = cache + 0.1
   call shmem_barrier_all()
   time1 = timef()

   . . . shmem code to be timed . . .

   time2 = timef()
   array_time(k) = time2 - time1
   call shmem_barrier_all()
end do
call shmem_real8_max_to_all(time,array_time,ntest,0,0,p,pWrk8,pSync1)
```

The first call to `shmem_barrier_all` guarantees that all processors reach this point before they each call the wall-clock timer, `timef`. The second call to `shmem_barrier_all` is to make sure that no processor starts the next iteration (flushing the cache) until all processors have completed executing the *"shmem code to be timed"*. The test is executed `ntest` times and the values of the differences in times on each participating processor are stored in `array_time`. The call to `shmem_real8_max_to_all` calculates the maximum of `array_time(k)` over all participating processors for each fixed `k` and places this maximum in `time(k)` for all values of `k`. Thus, `time(k)` is the time to execute the test for the k-th trial.

The clock ticks for `timef()` are every 0.013 microseconds on the T3E and every 0.800 microseconds on the Origin. This is not accurate enough to time very short messages so we were forced to time the *"shmem code to be timed"* multiple times between the two calls to `timef()`. However, when a shmem routine is used on the Origin to get a block of elements, `a(i:i+j)` for example, from a remote processor, all of the cache lines containing these elements are also moved to the cache of the local processor. If one then repeats this operation, then the block of elements `a(i:i+j)` reside on the cache of the local processor and there is no need to move it from the distant processor. This makes all subsequent executions of the test execute much faster than the first time. To avoid this problem, for each repetition of the test we take blocks of `a` that are spaced far enough apart.

To illustrate the ideas in the above paragraph, consider the second test: the right shift. Suppose p processors are a ring where the right neighbor of processor i is processor $(i+1)$ modulo p for $i = 0$ to $p-1$. For this test each processor gets a message from its left neighbor. For an 8 byte message this could be written as:

```
parameter (n=1)
real*8 a(n), b(n)
. . .
do i = 0, iter-1
   call shmem_get8(b(1),a(1),n,modulo(myrank-1,p))
end do
```

where `myrank` is the rank of the executing processor and `modulo(myrank-1,p)` is the left neighbor from which `a(1)` is obtained and written into `b(1)`. However, on the Origin the first iteration will execute much slower than subsequent iterations. This problem is corrected by the following:

```
integer, parameter (iter=40, stride=16, n=1, ndim=iter*stride)
real*8 a(0:ndim-1), b(0:ndim-1)
. . .
do i = 0, iter-1
   call shmem_get8(b(i*stride),a(i*stride),n,modulo(myrank-1,p))
end do
```

The value of `iter` was chosen so that the total execution time was at least 40 microseconds. This translated to `iter=40` for all tests with 8 byte messages and stride was taken to be 16, the number of elements in a cache line on the Origin (the cache line size on the T3E is 8). Of course, the values in the array `time(1:ntest)` are all divided by `iter` so they reflect the time for a single iteration of the test.

On the T3E, all remote memory references use the E-registers which bypass the cache [4]. Thus, cache flushing and the above use of `stride` have little effect on the performance of

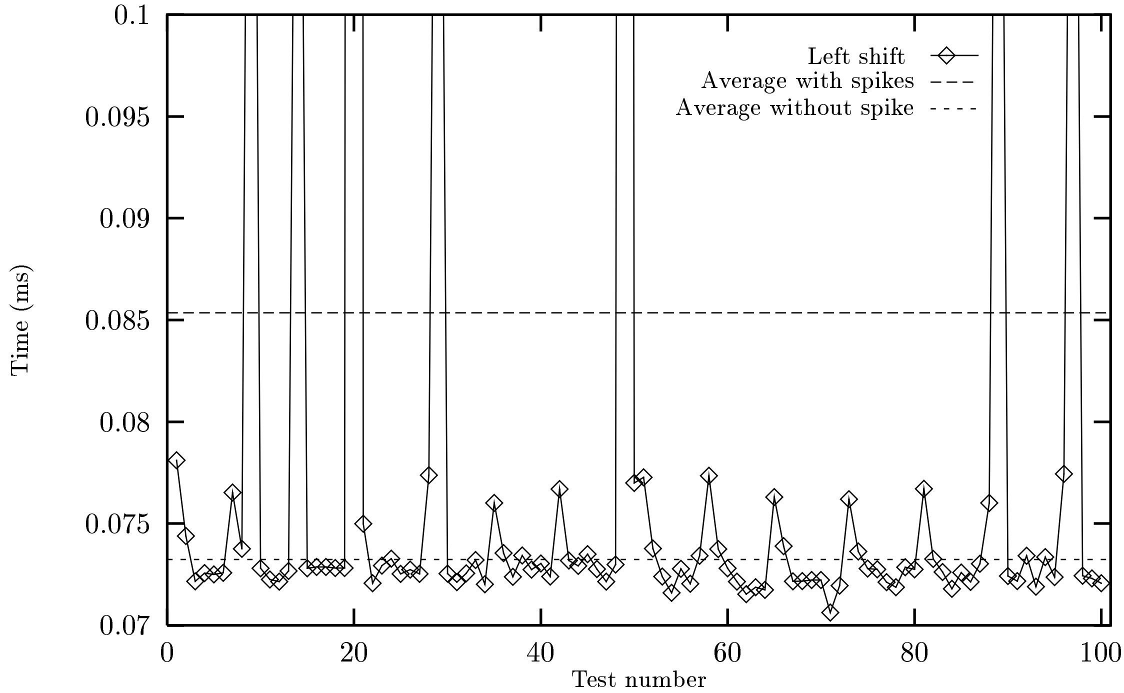

Figure 1: Execution times on the T3E for 100 executions of the right shift test with 68 processors and a 10 Kbyte message. Removing 7 spikes, the average decreases from about 0.085 microseconds to 0.073 microseconds.

these five tests on the T3E. However, exactly the same code was run on each machine for each test.

Usually, when a test is timed for the first time, there is additional overhead from calling the timer and other procedures for the first time that will not occur for subsequent calls. Because of this, the time for the first trial was always thrown out. This means that each run would produce 50 trials since the first trial would be thrown out (we use only the data of `time(2:ntest)` where `ntest=51`).

All tests were run twice making 100 trials for each test for a given message size and for a given number of processors. Figure 1 presents this data for the T3E for 68 processors and a 10 Kbyte message for the right shift. Notice that there are seven huge spikes (increases in time) which are likely due to operating system interruptions. The average of these 100 trials is approximately 0.085 microseconds (see the dotted line on the graph) which is much larger than most of the other 93 trials. It is the opinion of the authors that these spikes should be removed so that the performance data will reflect the times one will usually obtain. All data in this report have the spikes removed by the following process: First the median is computed and all times greater than 1.8 times this median are removed. There were a few cases where the above procedure would remove more than 10% of the data. The authors felt that this would not be appropriate, so in these cases the ten (10%) largest times were removed. An average was then calculated from the filtered data and this is what is presented for each test in this report. For the example in figure 3, the average of the filtered data is shown by the dotted line at about 0.073. For all tests and all message sizes, this filtering process led to the removal of an average of 0.43% of the trials for the T3E and 1.1% for the Origin 2000.

3 Communication tests and scalability results

Test 1: Accessing Distant Messages

This first test measures the time required for processor 0 to get a message from processor j for $j = 1$ to 127. The purpose is to determine the performance differences for sending messages between "close"processors and "distant" processors. In a perfectly scalable computer the time required to send a message would be independent of the processors used. The code to get a message of size `n` from processor `j` to processor `0` is:

```
if (myrank == 0) then
   call shmem_get8(a(i*stride),a(i*stride),n,j)
end if
```

where `myrank` is the rank of the processor executing this code.

The results of this test are based on one run per machine because the assignment of ranks to physical processors will vary from one run to another. Figure 2, page 7, contains the results of this test. For both machines, the time required to access large and small messages is roughly independent of the processors used. Notice that for the Origin, accessing messages from processors 6 and 7 is slower than accessing from other processors. This may be due to certain processes of the operating system that run on these processors [8]. Unlike the Origin, the T3E provides special additional processors dedicated to running the operating system. Also notice that on this test the "fat bristled hypercube" architecture of the Origin 2000 does not lead to a noticeable performance difference between communications inside a 32 processor hypercube and ones using the CrayLink Interconnect. In fact, none of the five tests in this paper show a noticeable performance difference whether or not communications are within a 32 processor hypercube.

Test 2: Right Shift

This test is described in section 2 and measures the performances when p processors get a message from their left neighbor, modulo p, for p = 4, 8, 12, ..., 128. The top graph in Figure 3, page 8, shows the ratio *(T3E Time)/ (Origin Time)* for each p. For 8 byte messages, the downward trend as p increases to 128 shows that the T3E scales better than the Origin 2000. For 1 Mbyte messages, neither machine significantly scales better than the other. The second and third graphs in figure 3 show the actual times for the execution of this test for the two message sizes.

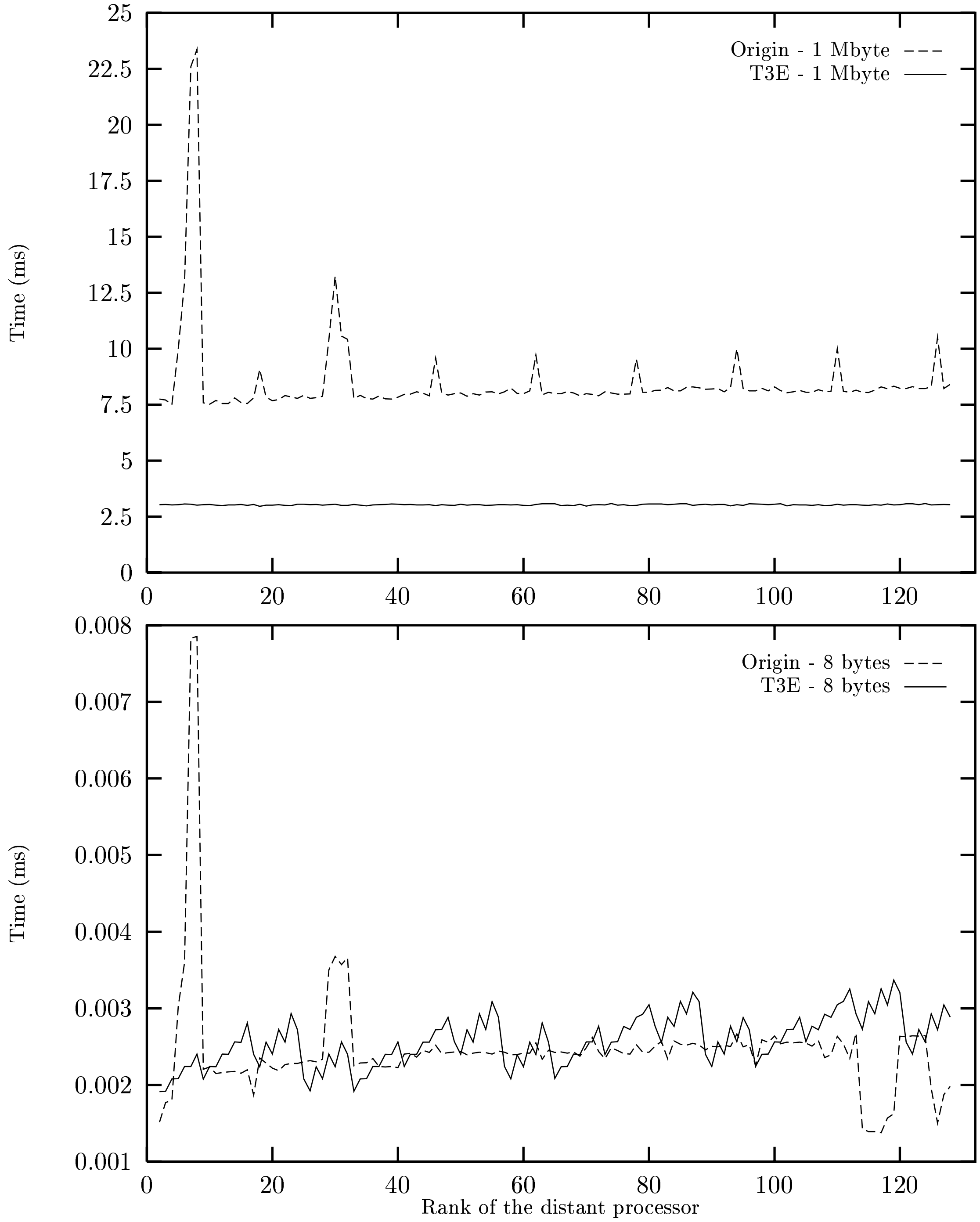

Figure 2: **Test 1: Accessing distant messages.** For clarity, only the lines connecting data points are shown.

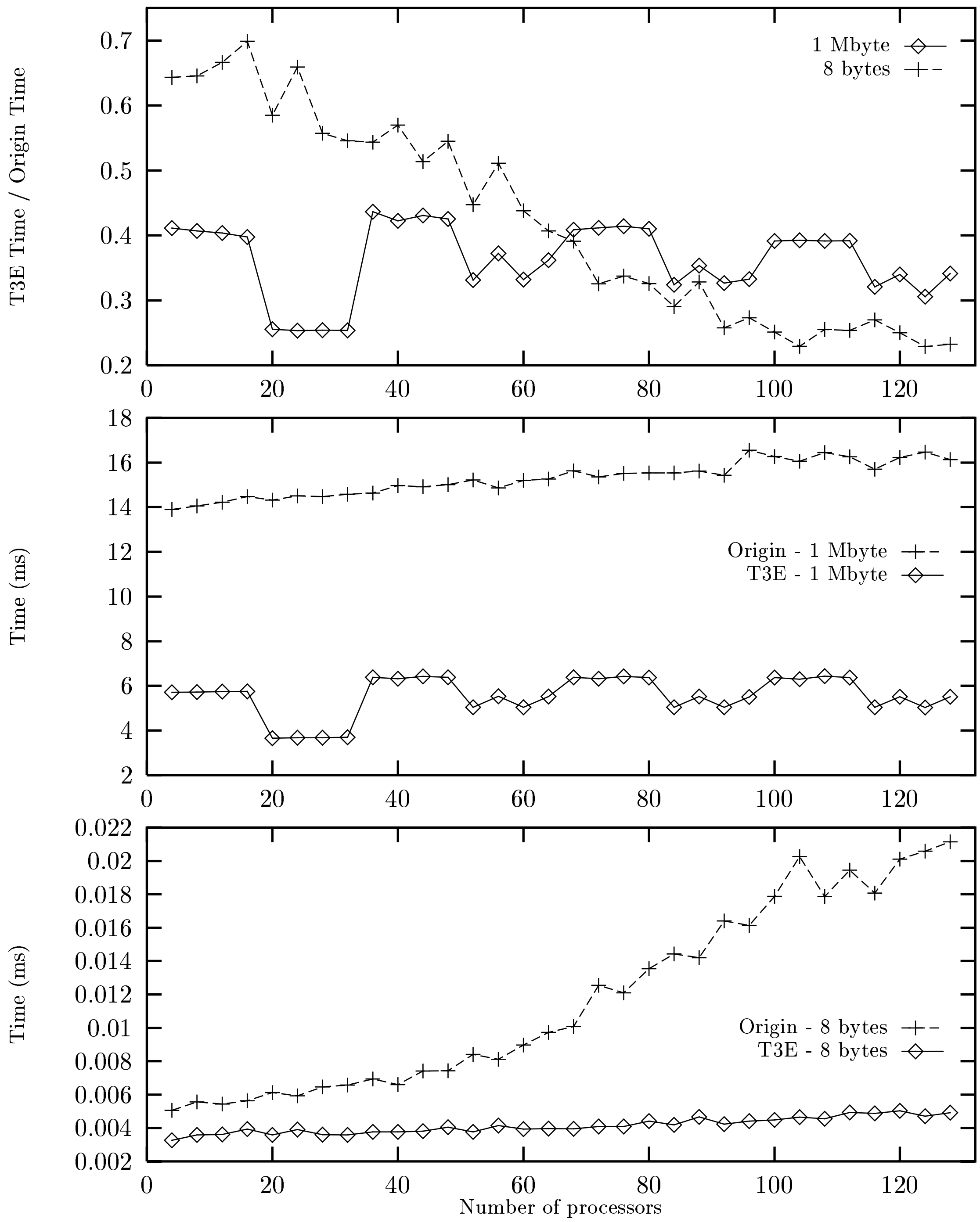

Figure 3: **Test 2: Right Shift**

Test 3: The Basic Broadcast

This test evaluates the ability for a processor to handle a large number of requests to get a message. This was accomplished by having all processors, except processor `0`, execute a `shmem_get8` for a message with `n` elements on processor `0`. The code used is:

```
if ((0 < myrank) .AND. (myrank < p)) then
   call shmem_get8(a(i*stride),a(i*stride),n,0)
end if
```

The top graph in figure 4, page 10, presents the ratio *(T3E Time)/ (Origin Time)* for each p. As p increases to 128, these ratios roughly approaches a horizontal line which means that the machines scale roughly the same. Notice also that unlike the two previous tests, the Origin 2000 significantly outperforms the T3E as the number of processors increases to 128 for the 1 Mbyte message.

Test 4: The Binary Tree Broadcast

The goal of this test was to evaluate the relative scalability of both machines with communications via a binary tree broadcast. This communication pattern is often used to implement other common operations such as reduce, gather and scatter [5].

To implement this test, synchronization is needed to ensure the correctness of the broadcast and to ensure that the next broadcast does not begin until the first one has completed when timing the 8 byte message. The code for an arbitrary number of processors is listed in appendix A on page 15. When $p = 2^n$, the algorithm for broadcasting a message `a` can be described as follows:

1. Using `shmem_put8`, processor 0 posts a message to processor $p/2$ indicating that it is ready to have `a` read from its memory. Using `shmem_int8_wait`, processor $p/2$ waits for processor 0 to post this message and then uses `shmem_get8` to read `a` from processor 0's memory after receiving the posted message.

2. Next, processors 0 and $p/2$ post a message to processors $p/4$ and $3p/4$, respectively, indicating that they are ready to have `a` read from their memories. Processors $p/4$ and $3p/4$ wait for processors 0 and $p/2$ to post these messages using `shmem_int8_wait` before using `shmem_get8` to read `a` from the memories of processors 0 and $p/2$.

3. The above pattern is repeated until all processors have received `a`. Using a `shmem_int8_wait`, processor 0 waits for the last processors that receive `a` to post a message that they have received `a` before the next broadcast is started.

Figure 5, page 11, presents the performance data for this test. Unlike the basic broadcast test, the T3E is always faster than the Origin 2000. Moreover, both machines scale roughly the same as the number of processors increases to 128.

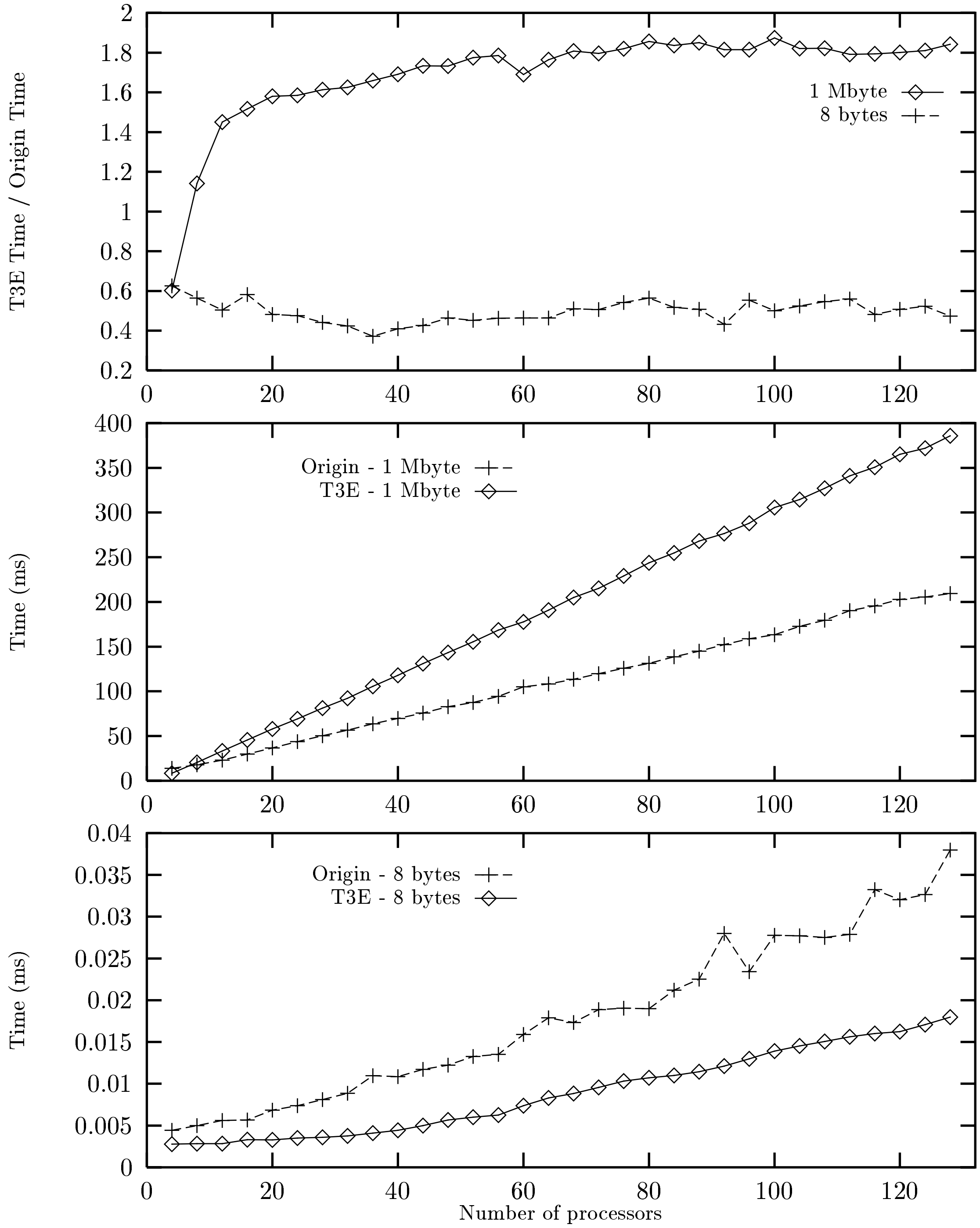

Figure 4: **Test 3: The Basic Broadcast**

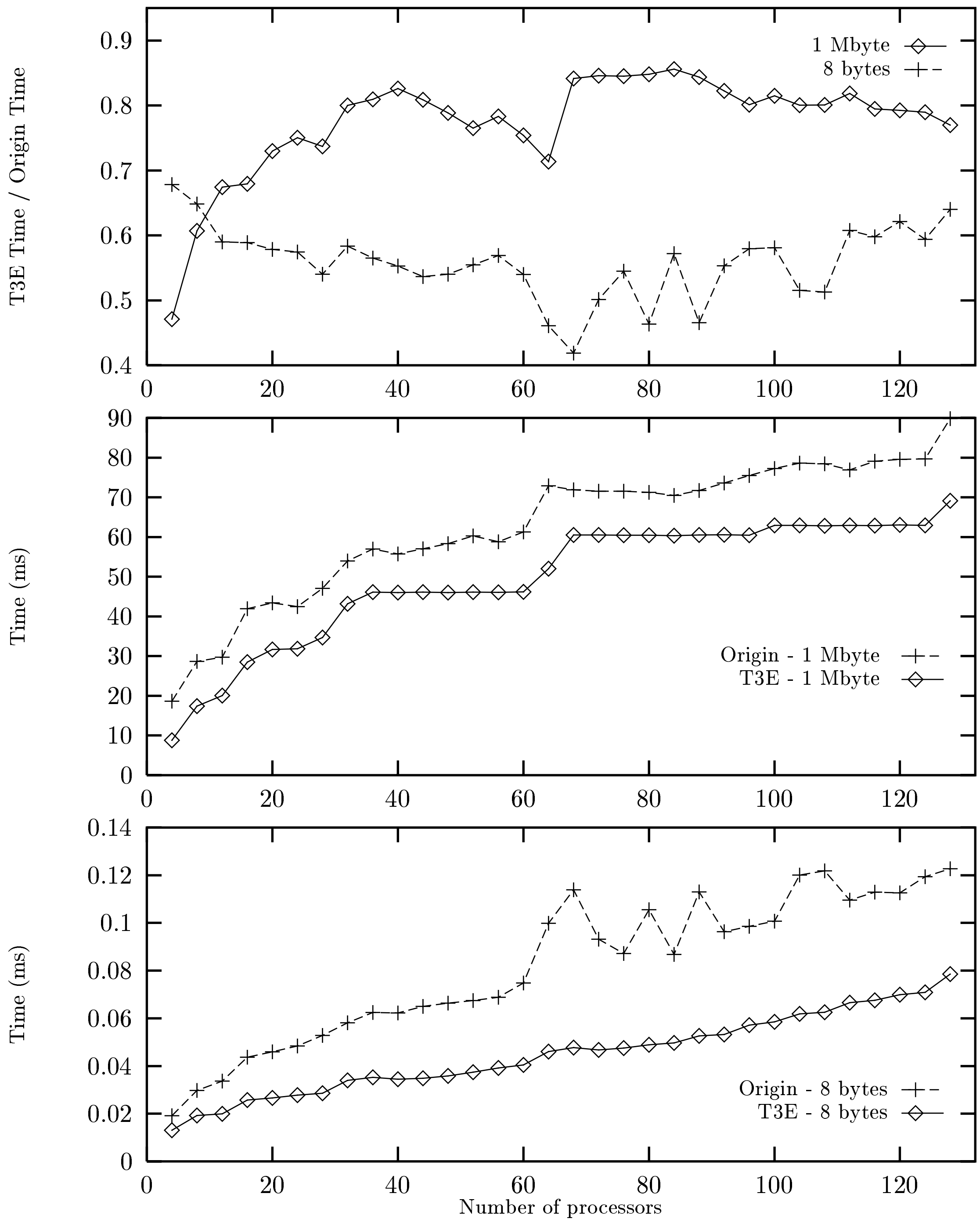

Figure 5: **Test 4: The Binary Tree Broadcast**

Test 5: The All-To-All

To evaluate the scalability of both machines for communication with high contention the all-to-all test was selected. For this test, each processor receives a message from each of the other processors. At step j, processor i gets a message from the processor $i+j$ modulo p, for $i=0$ to $p-1$. The code used is:

```
do j = myrank+1,p-1
   call shmem_get8(b(i*stride+n*j),a(i*stride+n*myrank),n,j)
end do
do j = 0,myrank-1
   call shmem_get8(b(i*stride+n*j),a(i*stride+n*myrank),n,j)
end do
```

Figure 6, page 13, presents the performance data for this test. For the 8 byte message the machines both scale about the same and the T3E is about 40% faster. For the 100 Kbyte message, the Origin 2000 is faster than the T3E and there is no significant difference between the scalability of both machines beyond 60 processors.

4 Conclusion

This study was conducted to evaluate relative scalability of the Cray T3E-600 and the Origin 2000. Five communication tests using SHMEM routines were chosen to represent commonly used communication patterns with low contention to communication patterns with high contention. The relative scalability was evaluated up to 128 processors, a limit imposed by the size of currently available Origin 2000 systems. Both machines scaled roughly the same on all tests except the right shift (test 2) with the 8 byte message where the T3E scaled significantly better than the Origin. However, it should be noted that the T3E outperformed the Origin 2000 for most of these tests.

5 Acknowledgments

We would like to thank SGI for allowing us to use their T3E-600 and Origin 2000 located in Eagan, Minnesota, USA.

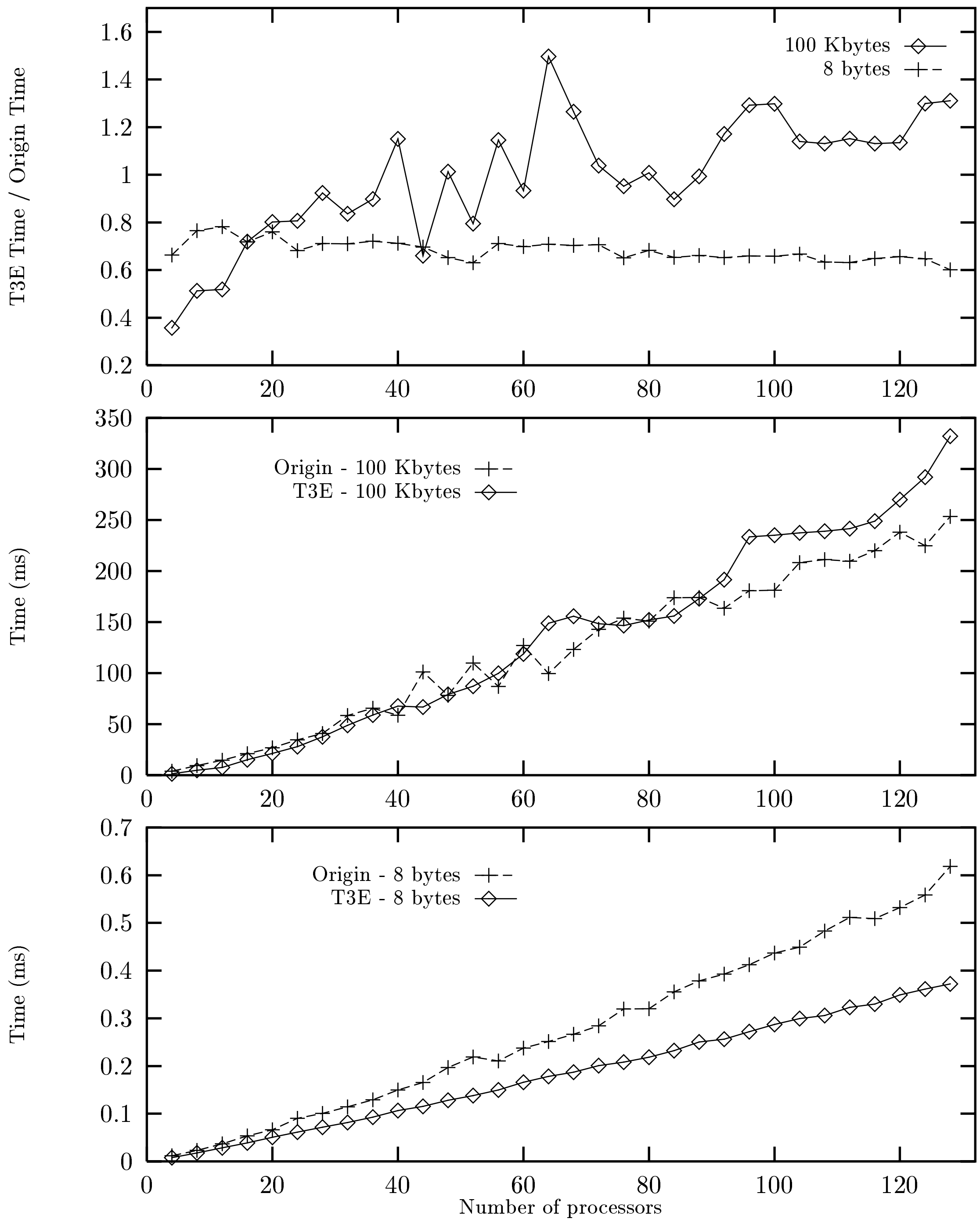

Figure 6: **Test 5: The All-To-All**

References

[1] Cray Research Web Server. http://www.cray.com.

[2] Origin Servers. Technical report, Silicon Graphics, April 1997.

[3] A. Anderson, J. Brooks, C. Grassl, and S. Scott. Performance of the CRAY T3E Multiprocessor. In *Proceedings of SC97*, 1997.

[4] E. Anderson, J. Brooks, and T. Hewitt. The Benchmarker's Guide to Single-processor Optimization for CRAY T3E Systems. Technical report, Benchmarking Group, Cray Research, June 1997.

[5] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Building a High-Performance Collective Communication Library. In *Supercomputing'94*, Washington D. C., November 1994. IEEE Computer Society Press.

[6] Cray Research. *Application Programmer Library Reference Manual.* Publication SR-2165.

[7] J. Fier. *Performance Tuning Optimization for Origin 2000 and Onyx 2.* Silicon Graphics, 1996. http://techpubs.sgi.com.

[8] J. M. D. Hill, S. R. Donaldson, and D. B. Skillicorn. Stability of Communication Performance in Practice: From the Cray T3E to Network of Workstations. Technical Report PRG-TR-33-97, Oxford University Computing Laboratory, Oxford, UK, 1997.

[9] G. R. Luecke and J. J. Coyle. Comparing the Performances of MPI on the Cray T3E-900, the Cray Origin 2000 and the IBM P2SC. *The Journal of Performance Evaluation and Modelling for Computer Systems*, June 1998. http://hpc-journals.ecs.soton.ac.uk/PEMCS/.

[10] M. Resch, H. Berger, T. Boenisch, and D. Sihling. Performance of MPI on Cray T3E-512. In *Third European CRAY-SGI MPP Workshop*, Paris, September 1997. http://armoise.saclay.cea.fr/~workshop/.

[11] S. L. Scott. Synchronization and Communication in the T3E Multiprocessor. In *ASPLOS VII.* ACM, October 1996. http://reality.sgi.com/sls_craypark/Papers/asplos96.html.

[12] S. L. Scott and G. M. Thorson. The Cray T3E Network: Adapatative Routing in a High Perfromance 3D Torus. In *HOT Interconnects*, Stanford University, August 1996.

[13] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI, The Complete Reference.* Scientific and Engineering Computation. The MIT Press, 1996.

A SHMEM code for the tree broadcast test

```
if (myrank == 0) then
   ! Send an acknowledgment to all targets
   target = 1
   do while(target <= p-1)
     target = ishft(target,+1)
   end do
   target = ishft(target,-1)
   do while(target>0)
     call shmem_put8(put_ok,YES,1,target)
     target = ishft(target,-1)
   end do
   ! Wait an end signal from each leave of the broadcast
   call shmem_int8_wait(end_bcast(p-1),NO)
   if (btest(p-1,0) .eqv.  .TRUE.) then
     target = p-3
   else
     target = p-2
   end if
   do while(target >0 )
     call shmem_int8_wait(end_bcast(target),NO)
     target = target-2
   end do
   end_bcast(1:p-1:2) = NO
   end_bcast(p-1) = NO
else
   ! Leaves of the brodcast tree
   if((btest(myrank,0) .eqv.  .TRUE.) .OR. (myrank == p-1)) then
     ! Compute the target rank
     position = 0
     do while(btest(myrank,position) .eqv.  .FALSE.)
       position = position+1
     end do
     ! Wait to receive an acknowledgment
     call shmem_int8_wait(put_ok,NO)
     put_ok = NO
     ! Read data
     call shmem_get8(a(i*stride),a(i*stride),n,ibclr(myrank,position))
     ! Signal end of broadcast to 0
     call shmem_put8(end_bcast(myrank),YES,1,0)
```

```
   else
      ! Intermediate processors
      ! Calculate target rank
      position = 1
      do while(btest(myrank,position) .eqv.  .FALSE.)
        position = position+1
      end do
      j = position-1
      target = ibset(myrank,j)
      do while(target > p-1)
        j = j-1
        target = ibset(myrank,j)
      end do
      ! Wait to receive an acknowledgment
      call shmem_int8_wait(put_ok,NO)
      put_ok = NO
      ! Read data
      call shmem_get8(a(i*stride),a(i*stride),n,ibclr(myrank,position))
      ! Send acknowledgment
      do while(j >= 0)
        call shmem_put8(put_ok,YES,1,ibset(myrank,j))
        j = j-1
      end do
   end if
end if
```