Measuring MPI Latency and Communication Rates for Small Messages

by
Glenn R. Luecke, Jim Coyle, Jim Hoekstra, and Marina Kraeva
grl, jjc, hoekstra, kraeva@iastate.edu
Iowa State University, Ames, Iowa 50011, USA
and
Ying Xu
yxu@ssc.net.cn
Shanghai Supercomputing Center, Shanghai, 201203, China
March 4, 2009

Abstract. The authors present a method for measuring MPI latency and communication rates for small messages. They first investigate the overhead and resolution of the wall-clock timer and then applying this to measure ping pong times with small numbers of ping pongs within each timing loop instead of using the traditional thousands of ping pongs within a timing loop. Since timings vary, standard statistical methods are used to analyze the timing data and the minimum, median, average and maximum timings are calculated as well as the standard deviation, standard error and relative standard error so one can understand the distribution of the timing data. The program that performs these calculations is freely available and can be downloaded by going to http://hpcgroup.public.iastate.edu/ .

1. Introduction

Currently, most scientific programs written for distributed memory parallel computers use the MPI message-passing library [1]. The speed that messages can be sent between processors using MPI routines is an important performance criterion for a parallel computer. Message passing performance between two nodes or within a node of a computer is commonly determined by measuring ping pong times when sending messages of various sizes. The operation of sending a message from one MPI process to another and back using blocking MPI sends and receives (i.e. mpi_send and mpi_recv) is usually called the ping pong operation. MPI latency is usually defined to be half of the time of a ping pong operation with a message of size zero.

The time to execute the ping pong operation for small messages on many of today's high performance communication networks is so small that the effects of the various software layers (e.g. OS, network card drivers, the software on the communication network) may cause ping pong times to vary significantly. Because of this, one should use standard statistical methods to describe the distribution of the timings such as minimum, mean, median, maximum, standard deviation, standard error and relative standard error. It is important to know the distribution of ping pong times and not just the average since large variations in ping pong times may result in poor load balancing and poor scalability for applications which spend significant amounts of time sending and receiving small messages.

Ideally, one would like an infinitely precise wall-clock timer to measure each ping pong time to obtain the distribution of ping pong times. However, on many of today's computers wall clock timers (e.g. mpi_wtime) are not able to accurately measure small ping pong times. For example, for the Opteron/InfiniPath cluster, named lightning, at Iowa State University (ISU), the resolution of mpi_wtime is about 1 microsecond and the average MPI ping pong latency times are around 1.3 microseconds (about 2.6 microseconds for a round trip). This lack of timer precision is usually

overcome by timing many (often thousands of) ping pongs within a single timing loop and then taking the average. However, taking an average of thousands of ping pongs within a single timing loop hides the distribution of the individual ping pong times. To better understand the distribution of ping pong times, the authors propose to not measure times with thousands of ping pongs in a single timing loop but to greatly reduce the number of ping pongs in a timing loop to a value based on the resolution of the timer and on a median ping pong time.

Communication rates between nodes depend on the size of the message being sent and are an important performance criterion of a parallel computer. For this study, uni-directional communication rates are measured using the ping pong operation with blocking MPI sends and receives. One can measure bi-directional communication rates by using either mpi_sendrecv or nonblocking MPI sends and receives [2]. Currently, the benchmarks in [2] do not perform multiple timing trials for statistical analysis.

A program has been written that automatically measures and statistically analyzes timing data for the ping pong operation using MPI blocking sends and receives. The program that performs these calculations is freely available and can be downloaded by going to http://hpcgroup.public.iastate.edu/ . Requests to have results posted on this website should be sent to Marina Kraeva at kraeva@iastate.edu .

2. Related Work

Currently, MPI latency and communication rates are usually measured using either the Ohio State University's OMB benchmarks [2] or Intel's MPI Benchmark [4] (formerly the PALLAS MPI test suite). The OMB benchmarks take an average of 10,000 ping pongs and the Intel benchmarks take an average of 1,000 ping pongs. Neither measures multiple timings for a single message size and no statistical analysis of timings is performed. We found it to be inconvenient that the OMB benchmark program is distributed with ping pongs between processors of rank 0 and 1. Intel's MPI Benchmark measures ping pong times between processors of rank 0 and p-1, where p is the number of MPI processes used when running the program. For the machines that the authors used, ping pongs between processors of rank 0 and p-1 were convenient because of the way MPI processes were distributed among nodes. For convenience, the program written for this paper measures ping pong times between MPI processes of rank "source" and "dest" where these values are set at the beginning of the program.

The seventh benchmark in the HPC Challenge Benchmark [3] is designed to measure the time required to perform simultaneous communication patterns and is based on the effective bandwidth benchmark, b_eff. The effective bandwidth benchmark measures the accumulated bandwidth of the communication network of parallel and/or distributed computing systems. Several message sizes, communication patterns and methods are used. The algorithm uses an average to take into account that short and long messages are transferred with different bandwidth values in real applications. These tests are not designed to measure latency and communication rates of small messages between two MPI processes.

3. Methodology

3.1 Measuring Timer Overhead and Resolution

Since timings are done with the MPI wall-clock timer, mpi_wtime, we first investigate the resolution and overhead of mpi_wtime. When timing ping pong operations, timings are performed on the processor from which the ping pong operation begins. The elapsed time of ping-pongs is measured in the standard way, i.e.

```
t1  =  mpi_wtime()
. . . ping-pongs …
t2  =  mpi_wtime()
time  =  t2 – t1
```

To investigate the resolution and overhead of mpi_wtime, we measure the times when there are no statements between timer calls, i.e.

```
do i  =  1,  n
    t1  =  mpi_wtime()
    t2  =  mpi_wtime()
    time(i) = t2 – t1
enddo
```

Experiments run on different machines showed a large variation in measured times so experiments were performed with a large number of timing trials to investigate the distribution of timings. With 2**24 timing trials, the following timing distribution was obtained using ISU's lightning cluster running RedHat Enterpise Linux 4.4 (Linux kernel 2.6.9-42.ELsmp). Times are in microseconds.

minimum time	=	0.95
median time	=	1.91
average time	=	1.62
maximum time	=	12500.05
standard deviation	=	1.61

Maximum times varied significantly from one run to another. For example, when making 10 runs with n = 2**24 the following maximum times were obtained: 161, 162, 266, 280, 327, 363, 723, 743, 891 and 12500 microseconds. The large variation in maximum times is likely due to interference from the operating system.

Average times over these same 10 runs varied from 1.62 to 1.75 microseconds. The minimum and median times were the same for all runs made (over 60 runs were made) when times were rounded as shown above. Appendix A shows the results of one of these runs and shows the complete distribution of all measured times.

When timing events of short duration, one should take the above timing overhead and timing variation into consideration. We therefore define the **resolution of the above timer, res_timing,** to be the minimum positive time of the times time(i), i = 1, n. Using the Fortran minval intrinsic function, this becomes:

res_timing = minval(time(1:n), mask = time(1:n) > 0.d0)

Experiments show that taking n = 2**24 was large enough to capture the minimum positive time. It is unlikely with this large value for n that all the measured times are zero, but if this does happen we define res_timing to be zero.

Define the **minimum timing overhead**, **min_ov,** to be the minimum of the non-negative times:

min_ov = minval(time(1:n), mask = time(1:n) >= 0.d0)

The minimum timing overhead is then subtracted from the elapsed time of "code-to-be-timed":

```
t1  =  mpi_wtime()
. . . code-to-be-timed …
t2  =  mpi_wtime()
time  =  t2 – t1 – min_ov
```

Notice that subtracting the average or median time instead of the minimum time might lead to some times being negative.

3.2 Timing Methodology

When measuring ping pong times the minimum timing overhead is subtracted from the measured times. This is applied to measuring "npp" ping pongs within a single timing loop as follows:

```
do ktrial = 1, ntrial
   call mpi_barrier(comm, ierror)
   if (rank == source) then
      t1 = mpi_wtime()
      do i = 1, npp
         call mpi_send(A(1), n, dp, dest , 1, comm, ierror)
         call mpi_recv(A(1), n, dp, dest, 1, comm, mpi_status_ignore, ierror)
      enddo
      t2 = mpi_wtime()
      time(ktrial) = t2 – t1 - min_ov
   endif
   if (rank == dest) then
      do i = 1, npp
         call mpi_recv(A(1), n, dp, source, 1, comm, mpi_status_ignore, ierror)
         call mpi_send(A(1), n, dp, source, 1, comm, ierror)
      enddo
   endif
enddo
```

where "dp" is "mpi_double_precision" and the ping pongs are between MPI processes of rank "source" and "dest".

Notice that the source process may release from mpi_barrier before the dest process. This can cause unnecessary variability in the measured timing data. To lessen this variability, a hand-shaking procedure was added by calling mpi_recv before the call to the timer on the rank = source process and calling mpi_send on the rank = dest process as follows: (The two additional hand-shaking statements are highlighted in bold.)

```
do ktrial = 1, ntrial
    call mpi_barrier(comm, ierror)
    if (rank == source) then
       call mpi_recv(A(1), 1, dp, dest, 1, comm, mpi_status_ignore, ierror)
       t1 = mpi_wtime()
       do i = 1, npp
          call mpi_send(A(1), n, dp, dest , 1, comm, ierror)
          call mpi_recv(A(1), n, dp, dest, 1, comm, mpi_status_ignore, ierror)
       enddo
       t2 = mpi_wtime()
       time(ktrial) = t2 – t1 - min_ov
    endif
    if (rank == dest) then
       call mpi_send(A(1), 1, dp, source, 1, comm, ierror)
       do i = 1, npp
          call mpi_recv(A(1), n, dp, source, 1, comm, mpi_status_ignore, ierror)
          call mpi_send(A(1), n, dp, source, 1, comm, ierror)
       enddo
    endif
enddo
```

Notice that this hand-shaking does not guarantee that the MPI receive is posted before the ping pong operation begins. For example, it could happen that a long interruption occurs right after the mpi_send is executed on the processor of rank = dest. However, experiments show that using this hand-shaking does lower the variability of the timing data.

3.3 Estimating npp - the number of ping pongs in a timing loop

Let npp be the number of ping pongs within each timing loop. Ideally one would like to precisely measure the time of a single ping pong many times and then statistically analyze the timing data. However, this is not possible due to the lack of timer precision, especially when ping pong times are of the same order as the resolution of the timer. Thus, we have the following dilemma:

- If npp is large, then we can accurately measure timing trials, but the information about the variability of each timing trial is unknown.
- If npp is small, then the information about the variability of the timing trials is known, but these times cannot be measured accurately.

We now present a method for choosing npp in a way that takes into account both the resolution of the timer and a median time for the ping pong. To illustrate this method, suppose the resolution of the timer is 1 microsecond and that the time for a single, round trip ping pong time is 2 microseconds. For this case, we suggest choosing npp to be 25 so that there will be about 50 = 50*(res_timing) microseconds in each timing loop. There was much discussion among the authors regarding whether we should recommend a factor of 10, 25, 50, 100 or something else. We also discussed choosing npp so the standard deviation for the timing trials was less than some value. In the end, the authors decided to recommend that npp be large enough so that timings were “accurate” within 50*(res_timing). In the program this factor is named **res_npp** and it can be easily set to any positive integer.

The MPI API says that the MPI function, mpi_wtick, is to return the value of the clock tick for the MPI wall-clock timer, mpi_wtime. Initially, we considered using the value of the clock tick to approximate

npp. However, we could not use mpi_wtick since we discovered that the value returned by mpi_wtick was not constant on some machines.

The estimated value for npp is calculated as follows. Let npp_init be an initial value given npp to calculate an approximation for a single ping pong time. Let the double precision array npptime(1:npptrial) be the measured times for "npptrial" timings with npp_init ping pongs within a single timing loop. We now need to decide whether to use the minimum, median or average of these times to calculate an approximation for the ping pong time. Experiments show that the median time is the most stable when making multiple timing trials, so this is what is used. Then the median time for one ping pong, median_ppt is

median_ppt = median(npptime(1:npptrial)/dble(npp_init))

Using res_timing defined above, one would like

npp*median_ppt >= (res_npp)*(res_timing)

Dividing the above inequality by median_ppt gives

npp >= (res_npp)*(res_timing)/median_ppt

It might happen that the right hand side of the above inequality is less than 1.d0. When this happens, npp should have the value of 1. Thus,

npp = max(1.d0, (res_npp)*(res_timing)/median_ppt))

Since the right hand side of the above equation may not have an integer value and since npp must be a positive integer, we use the nearest integer intrinsic function, nint, and define

npp = nint(max(1.d0, (res_npp)*(res_timing)/median_ppt))

Occasionally the computed value of npp may change from one run to another due to small variations in the values of res_timing and/or median_ppt. To be able to compare timing data from one run to another, one should do this with the same value of npp. Thus (as has already been mentioned), this program has the option of setting the value of npp to be any positive integer instead of having its value calculated. This option also allows one to explore the effects of trying different values of npp.

3.4 Analysis of Ping Pong Timing Data

After the ping pong timing data has been collected, it is then analyzed using standard statistical techniques [5]. Specifically, in addition to calculating the minimum, median, average and maximum times, we also calculate the variance, standard deviation, coefficient of variation, standard error, relative standard error (i.e. the standard error divided by the average) and the distribution of the timing data. Communication rates using the minimum, median and average times are also calculated. Appendix B shows the distribution of ping pong times from the lightning Opteron/InfiniPath cluster at Iowa State University when npp = 1. Appendix C gives the summary output from a run on the lightning Opteron/InfiniPath cluster at Iowa State University with npp = 18.

When there are a large number of times much greater than the median time, then these will increase the average value. Statistically these large values could be considered as "outliers" that should be filtered out. The program written by the authors not only statistically analyzes the unfiltered timing data but

does this same analysis on the filtered timing data when removing all timing trials that are larger than 2 times the median. This cut off value is set via the "cut_coef" and can be easily changed. Filtering the timing data stabilizes the average value, but does not allow one to know the variability of the timing data.

4. Results

4.1 Lightning timing results using the InfiniPath/HTX interconnect

Recall that ntrial is the number of timing trials and that npp is the number of ping pongs in a timing loop. Table 4.1 compares the timing results for a message of size 8 bytes when running on the lightning cluster with the InfiniPath/HTC interconnect with ntrial at least 64 million when npp = 1, 9, 18, 36, 72 and 144. When npp = 15000, ntrial was 15000. The values of npp = 9, 18, 36, 72 and 144 correspond to the calculated value of npp when res_npp = 25, 50 (the recommended value), 100, 200 and 400, respectively. npp = 15000 was chosen since standard latency benchmarks time 10 to 15 thousand ping pongs and measure a single time instead of making multiple timings with a timing loop.

npp =	1	9	18	36	72	144	15000
minimum time	1.43	1.22	1.25	1.26	1.36	1.36	1.37
median time	2.03	1.34	1.31	1.31	1.40	1.38	1.38
average time	1.81	1.36	1.32	1.32	1.40	1.39	1.38
maximum time	306.96	1133.88	570.81	284.50	45.79	26.06	2.43
Variance	0.21	0.27	0.15	0.06	0.0037	0.0022	0.00010
Standard deviation	0.46	0.52	0.38	0.24	0.060	0.046	0.010

Table 4.1. Lightning using the HTX interconnect: ping pong times in microseconds divided by 2 for 8 byte messages.

Notice that as npp increases, the variability of the measured times decreases and this hides the variability of the ping pong times. The variability of the timing data is important to know since this may affect program load balancing and scalability. Notice that even when npp = 15,000 the measured times vary from 1.37 to 2.43 microseconds. For the lighting cluster, the resolution of the timer is about 1 microsecond and the median (round trip) ping pong time is about 2.6 microseconds, so the accuracy of the median and average times is questionable when npp = 1.

Appendix B shows the distribution of measured times divided by 2 for the npp = 1. Notice that one can see the granularity of the timer, mpi_wtime. Appendix C shows the summary output file produced when npp = 18.

4.2 Lightning timing results using the Gigabit Ethernet interconnect

Table 4.2 compares the timing results for a message of size 8 bytes when running on the lightning cluster using the gigabit Ethernet interconnect with ntrial large enough so the execution time of the program takes about one hour. For example, when npp=1, ntrial =2**20 and when npp=15000, ntrial=2**10. The calculated value of npp was 1 with res_npp = 50. The gigabit Ethernet switch consists of four 48-port stackable switches. Tests were run on compute nodes connected to adjacent ports on a single 48-port switch.

npp =	1	4	8	16	32	128	15000
minimum time	46.5	48.9	52.6	53.3	53.5	54.6	54.6
median time	61.0	63.1	55.7	55.4	55.6	55.4	55.0
average time	61.9	62.8	57.0	56.2	55.9	56.4	55.1
maximum time	1433.5	647.6	2727.4	1334.2	313.8	225.7	60.6
Variance	78.8	51.9	44.0	22.9	5.6	8.4	0.13
standard deviation	8.9	7.7	6.6	4.8	2.4	2.9	0.36

Table 4.2. Lightning using the Gigabit Ethernet interconnect: ping pong times in microseconds divided by 2 for 8 byte messages.

As was the case in Table 4.1, as npp increases the variability of the measured times decreases hiding the variability of the ping pong times. Notice that even when npp = 15000 the measured times vary from 54.6 to 60.6 microseconds.

4.3 SSC's Magic Cube timing results using an InfiniBand interconnect

The Magic Cube computer at the Shanghai Supercomputing Center (SSC) consists of 1500 nodes with each node having four AMD quad core, 1.9GHz Opteron (8347) processors sharing 64 GB of memory. Nodes are interconnected with an InfiniBand switch. The InfiniBand (IB) switch has two levels with every 10 compute nodes connected to a 24 port IB switch and the 24 port IB switches connected to upper level IB switches. Timings were measured between 2 nodes on a single 24 port IB switch. Magic Cube is running SuSe SLSE 10 SP2 and the compiler used is gcc version 4.1.2 20070115 (SUSE Linux) and gfortran 4.1.2 (SUSE Linux). MPI used is MVAPICH 1.1 compiled with gcc.

Table 4.3 compares the timing results for a message of size 8 bytes when running on Magic Cube using the InfiniBand interconnect with ntrial large enough so the execution time of the program takes about one hour. For example, when npp=1, ntrial =2**20 and when npp=15000, ntrial=2**10. The calculated the value of npp was 16 with res_npp = 50.

npp =	1	8	16	32	64	128	15000
minimum time	2.00	1.69	1.66	1.63	1.67	1.70	1.73
median time	2.50	1.81	1.81	1.80	1.76	1.75	1.76
average time	2.35	1.82	1.82	1.79	1.76	1.75	1.76
maximum time	358.50	340.06	362.22	311.86	74.54	37.39	2.63
Variance	0.31	0.17	0.14	0.12	0.011	0.006	0.0005
standard deviation	0.56	0.40	0.37	0.35	0.10	0.078	0.022

Table 4.3. Magic Cube with InfiniBand: ping pong times in microseconds divided by 2 for 8 byte messages.

As was the case in Table 4.1, as npp increases the variance and standard deviation of the measured times decreases hiding the variability of the ping pong times.

5. Conclusions

The authors present a method for measuring MPI latency and communication rates for small messages. When investigating the overhead and resolution of the timer, they found large variations in measured times when there were no statements between consecutive timer calls. Using this timing information, the authors present a method for timing ping pong operations that uses a small number of ping pongs within a timing loop instead of the traditional methods that use thousands of ping pongs in a timing loop. The small number of ping pongs in a timing loop is determined by the resolution of the timer and by an approximation to the time of a single ping pong. Having a small number of ping pongs in a timing loop allows one to better understand the variability of ping pong times. The distribution of timing data is calculated and then analyzed using standard statistical methods.

A program has been written that automatically measures and statistically analyzes timing data for the ping pong operation. The program allows one to select any value for the number of timing trials and the number of ping pongs in the timing loop. When the program is run, two files are produced: latency.output and latency.summary.output. The first file contains all the output produced and the second is a summary of the results. The program that performs these calculations is freely available and can be downloaded by going to http://hpcgroup.public.iastate.edu/ . Requests to have results posted on this website should be sent to Marina Kraeva at kraeva@iastate.edu .

References

1. 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.
2. The Ohio State University Benchmarks: http://nowlab.cse.ohio-state.edu/projects/mpi-iba/benchmarks.html
3. The HPC Challenge Benchmark 7: Communication Bandwidth and Latency. http://icl.cs.utk.edu/hpcc/index.html
4. Intel's MPI Benchmarks 3.1: http://www.intel.com/cd/software/products/asmo-na/eng/cluster/clustertoolkit/219848.htm
5. Snedecor, George W. and Cochran, William G. (1989), Statistical Methods, Eighth Edition, Iowa State University Press.

Appendix A: Distribution of timing in time(1:n) on the ISU's lightning Opteron/InfiniPath cluster when running the following with n = 2**24

```
do i  =  1, n
   t1    =  mpi_wtime()
   t2  =  mpi_wtime()
   time(i) = t2 – t1
 enddo
```

DISTRIBUTION OF TIMES FROM THE TIMER OVERHEAD MEASUREMENTS IN PART 1:

```
0.00  <=  times  <  0.20 with count   =        0
0.20  <=  times  <  0.40 with count   =        0
0.40  <=  times  <  0.60 with count   =        0
0.60  <=  times  <  0.80 with count   =        0
0.80  <=  times  <  1.00 with count   =   5118946
1.00  <=  times  <  1.20 with count   =   1234766
1.20  <=  times  <  1.40 with count   =        0
1.40  <=  times  <  1.60 with count   =        0
1.60  <=  times  <  1.80 with count   =        0
```

1.80 <= times < 2.00 with count = 6351041
2.00 <= times < 2.20 with count = 4035896
2.20 <= times < 2.40 with count = 0
2.40 <= times < 2.60 with count = 0
2.60 <= times < 2.80 with count = 0
2.80 <= times < 3.00 with count = 3665
3.00 <= times < 3.20 with count = 5052
3.20 <= times < 3.40 with count = 0
3.40 <= times < 3.60 with count = 0
3.60 <= times < 3.80 with count = 0
3.80 <= times < 4.00 with count = 6
4.00 <= times < 4.20 with count = 37
4.20 <= times < 4.40 with count = 0
4.40 <= times < 4.60 with count = 0
4.60 <= times < 4.80 with count = 1
4.80 <= times < 5.00 with count = 0
5.00 <= times < 5.20 with count = 13
5.20 <= times < 5.40 with count = 0
5.40 <= times < 5.60 with count = 0
5.60 <= times < 5.80 with count = 0
5.80 <= times < 6.00 with count = 222
6.00 <= times < 6.20 with count = 46
6.20 <= times < 6.40 with count = 0
6.40 <= times < 6.60 with count = 0
6.60 <= times < 6.80 with count = 0
6.80 <= times < 7.00 with count = 997
7.00 <= times < 7.20 with count = 522
7.20 <= times < 7.40 with count = 0
7.40 <= times < 7.60 with count = 0
7.60 <= times < 7.80 with count = 0
7.80 <= times < 8.00 with count = 1291
8.00 <= times < 8.20 with count = 1600
8.20 <= times < 8.40 with count = 0
8.40 <= times < 8.60 with count = 0
8.60 <= times < 8.80 with count = 0
8.80 <= times < 9.00 with count = 242
9.00 <= times < 9.20 with count = 732
9.20 <= times < 9.40 with count = 0
9.40 <= times < 9.60 with count = 0
9.60 <= times < 9.80 with count = 10
9.80 <= times with count = 22131

FREQUENCY OF TIMES FROM THE TIMER OVERHEAD MEASUREMENTS IN PART 1:
time = 0.954 with frequency = 5118946
time = 1.192 with frequency = 1234766
time = 1.907 with frequency = 6351041
time = 2.146 with frequency = 4035896
time = 2.861 with frequency = 3665
time = 3.099 with frequency = 5052
time = 3.815 with frequency = 6
time = 4.053 with frequency = 37
time = 4.768 with frequency = 1
time = 5.007 with frequency = 13
time = 5.960 with frequency = 222
time = 6.199 with frequency = 46

time = 6.914 with frequency = 997
time = 7.153 with frequency = 522
time = 7.868 with frequency = 1291
time = 8.106 with frequency = 1600
time = 8.821 with frequency = 242
time = 9.060 with frequency = 732
time = 9.775 with frequency = 10
time = 10.014 with frequency = 210
time = 10.967 with frequency = 10
time = 11.206 with frequency = 2
time = 12.159 with frequency = 4
time = 12.875 with frequency = 45
ime = 13.113 with frequency = 43
time = 13.828 with frequency = 457
time = 14.067 with frequency = 1128
time = 14.782 with frequency = 507
time = 15.020 with frequency = 5627
time = 15.974 with frequency = 8030
time = 16.212 with frequency = 971
time = 16.928 with frequency = 3163
time = 17.166 with frequency = 1328
time = 17.881 with frequency = 173
time = 18.120 with frequency = 193
time = 18.835 with frequency = 6
time = 19.073 with frequency = 14
time = 19.789 with frequency = 1
time = 20.027 with frequency = 3
time = 20.981 with frequency = 4
time = 21.935 with frequency = 1
time = 22.888 with frequency = 8
time = 23.127 with frequency = 5
time = 23.842 with frequency = 9
time = 24.080 with frequency = 9
time = 24.796 with frequency = 1
time = 25.034 with frequency = 10
time = 25.988 with frequency = 1
time = 29.087 with frequency = 2
time = 29.802 with frequency = 2
time = 30.041 with frequency = 4
time = 30.994 with frequency = 14
time = 31.948 with frequency = 6
time = 32.187 with frequency = 1
time = 32.902 with frequency = 4
time = 33.140 with frequency = 3
time = 33.855 with frequency = 1
time = 34.094 with frequency = 5
time = 34.809 with frequency = 1
time = 35.048 with frequency = 2
time = 36.001 with frequency = 4
time = 36.955 with frequency = 1
time = 38.862 with frequency = 2
time = 40.054 with frequency = 1
time = 41.008 with frequency = 4
time = 41.962 with frequency = 12
time = 42.200 with frequency = 2

time = 42.915 with frequency = 5
time = 43.154 with frequency = 5
time = 43.869 with frequency = 3
time = 44.107 with frequency = 3
time = 44.823 with frequency = 2
time = 45.061 with frequency = 4
time = 46.015 with frequency = 3
time = 47.922 with frequency = 2
time = 48.876 with frequency = 1
time = 49.829 with frequency = 1
time = 51.022 with frequency = 1
time = 57.936 with frequency = 1
time = 77.963 with frequency = 1
time = 78.917 with frequency = 1
time = 82.970 with frequency = 1
time = 83.923 with frequency = 1
time = 84.877 with frequency = 2
time = 85.115 with frequency = 1
time = 86.069 with frequency = 3
time = 87.023 with frequency = 1
time = 87.976 with frequency = 2
time = 89.169 with frequency = 2
time = 89.884 with frequency = 1
time = 93.937 with frequency = 1
time = 97.036 with frequency = 1
time = 108.957 with frequency = 7
time = 109.911 with frequency = 4
time = 110.149 with frequency = 4
time = 110.865 with frequency = 1
time = 114.918 with frequency = 1
time = 116.110 with frequency = 1
time = 117.064 with frequency = 1
time = 119.925 with frequency = 1
time = 120.878 with frequency = 1
time = 121.117 with frequency = 1
time = 121.832 with frequency = 2
time = 123.024 with frequency = 1
time = 123.978 with frequency = 1
time = 124.931 with frequency = 1
time = 126.123 with frequency = 1
time = 128.031 with frequency = 2
time = 130.892 with frequency = 2
time = 133.038 with frequency = 3
time = 133.991 with frequency = 3
time = 134.945 with frequency = 1
time = 137.091 with frequency = 1
time = 140.905 with frequency = 1
time = 162.125 with frequency = 1
time = 190.973 with frequency = 1
time = 209.808 with frequency = 1
time = 211.000 with frequency = 1
time = 227.928 with frequency = 2
time = 1204.967 with frequency = 1

Appendix B: The following distribution of ping pong timing data is from the lightning Opteron/InfiniPath cluster at Iowa State University and is the distribution of measured times divided by 2 for the npp=1 case. Times are in microseconds and the number of timing trials = 2**25.

MEASURED TIMINGS AND THEIR FREQUENCY IN MICROSECONDS

time =	1.431	with frequency =	3480329
time =	1.550	with frequency =	12141037
time =	1.907	with frequency =	503323
time =	2.027	with frequency =	17183520
time =	2.503	with frequency =	48120
time =	2.623	with frequency =	9500
time =	2.980	with frequency =	8135
time =	3.099	with frequency =	4482
time =	3.457	with frequency =	6653
time =	3.576	with frequency =	8490
time =	3.934	with frequency =	5486
time =	4.053	with frequency =	16329
time =	4.411	with frequency =	1269
time =	4.530	with frequency =	21748
time =	5.007	with frequency =	14071
time =	5.126	with frequency =	2234
time =	5.484	with frequency =	7322
time =	5.603	with frequency =	3662
time =	5.960	with frequency =	5201
time =	6.080	with frequency =	5525
time =	6.437	with frequency =	2526
time =	6.557	with frequency =	6455
time =	6.914	with frequency =	979
time =	7.033	with frequency =	10464
time =	7.510	with frequency =	9793
time =	7.629	with frequency =	1168
time =	7.987	with frequency =	11452
time =	8.106	with frequency =	4867
time =	8.464	with frequency =	9736
time =	8.583	with frequency =	9707
time =	8.941	with frequency =	2790
time =	9.060	with frequency =	6488
time =	9.418	with frequency =	24
time =	9.537	with frequency =	187
time =	10.014	with frequency =	61
time =	10.133	with frequency =	7
time =	10.490	with frequency =	55
time =	10.610	with frequency =	25
time =	10.967	with frequency =	38
time =	11.086	with frequency =	36
time =	11.444	with frequency =	11
time =	11.563	with frequency =	19
time =	11.921	with frequency =	5
time =	12.040	with frequency =	17
time =	12.517	with frequency =	29
time =	12.636	with frequency =	3
time =	12.994	with frequency =	64
time =	13.113	with frequency =	24
time =	13.471	with frequency =	33
time =	13.590	with frequency =	30

time = 13.947 with frequency = 19
time = 14.067 with frequency = 42
time = 14.424 with frequency = 10
time = 14.544 with frequency = 29
time = 15.020 with frequency = 16
time = 15.497 with frequency = 33
time = 15.616 with frequency = 10
time = 15.974 with frequency = 32
time = 16.093 with frequency = 23
time = 16.451 with frequency = 29
time = 16.570 with frequency = 37
time = 16.928 with frequency = 8
time = 17.047 with frequency = 45
time = 17.405 with frequency = 1
time = 17.524 with frequency = 49
time = 18.001 with frequency = 39
time = 18.120 with frequency = 13
time = 18.477 with frequency = 18
time = 18.597 with frequency = 15
time = 18.954 with frequency = 7
time = 19.073 with frequency = 8
time = 19.431 with frequency = 1
time = 19.550 with frequency = 2
time = 20.027 with frequency = 9
time = 20.504 with frequency = 3
time = 20.623 with frequency = 2
time = 20.981 with frequency = 2
time = 21.100 with frequency = 1
time = 21.458 with frequency = 1
time = 21.577 with frequency = 1
time = 22.054 with frequency = 5
time = 22.411 with frequency = 1
time = 22.531 with frequency = 3
time = 23.007 with frequency = 4
time = 23.127 with frequency = 1
time = 23.484 with frequency = 1
time = 23.603 with frequency = 2
time = 23.961 with frequency = 1
time = 24.557 with frequency = 1
time = 25.034 with frequency = 2
time = 25.988 with frequency = 1
time = 27.061 with frequency = 2
time = 27.537 with frequency = 4
time = 28.014 with frequency = 1
time = 28.491 with frequency = 1
time = 28.968 with frequency = 3
time = 29.445 with frequency = 1
time = 29.564 with frequency = 1
time = 30.041 with frequency = 2
time = 30.518 with frequency = 4
time = 31.471 with frequency = 2
time = 32.425 with frequency = 1
time = 33.021 with frequency = 1
time = 34.451 with frequency = 1
time = 35.048 with frequency = 6

time = 35.524 with frequency = 8
time = 36.001 with frequency = 5
time = 36.120 with frequency = 2
time = 36.478 with frequency = 5
time = 36.597 with frequency = 7
time = 36.955 with frequency = 5
time = 37.074 with frequency = 7
time = 37.551 with frequency = 8
time = 38.028 with frequency = 11
time = 38.505 with frequency = 4
time = 38.981 with frequency = 4
time = 39.101 with frequency = 4
time = 39.458 with frequency = 6
time = 39.577 with frequency = 7
time = 39.935 with frequency = 1
time = 40.054 with frequency = 6
time = 40.412 with frequency = 1
time = 40.531 with frequency = 6
time = 41.008 with frequency = 2
time = 41.962 with frequency = 2
time = 42.439 with frequency = 2
time = 42.558 with frequency = 3
time = 43.035 with frequency = 4
time = 43.511 with frequency = 6
time = 43.631 with frequency = 1
time = 43.988 with frequency = 6
time = 44.107 with frequency = 4
time = 44.465 with frequency = 5
time = 44.584 with frequency = 2
time = 44.942 with frequency = 1
time = 45.061 with frequency = 4
time = 45.538 with frequency = 4
time = 46.015 with frequency = 1
time = 46.492 with frequency = 2
time = 48.518 with frequency = 1
time = 48.995 with frequency = 2
time = 49.472 with frequency = 1
time = 51.022 with frequency = 1
time = 51.498 with frequency = 1
time = 52.452 with frequency = 1
time = 52.571 with frequency = 1
time = 53.525 with frequency = 2
time = 54.002 with frequency = 1
time = 54.479 with frequency = 8
time = 54.598 with frequency = 2
time = 54.955 with frequency = 1
time = 55.075 with frequency = 4
time = 55.432 with frequency = 2
time = 55.552 with frequency = 8
time = 55.909 with frequency = 3
time = 56.028 with frequency = 14
time = 56.505 with frequency = 5
time = 56.624 with frequency = 3
time = 56.982 with frequency = 12
time = 57.101 with frequency = 2

time = 57.459 with frequency = 1
time = 57.578 with frequency = 7
time = 57.936 with frequency = 6
time = 58.055 with frequency = 4
time = 58.413 with frequency = 1
time = 58.532 with frequency = 9
time = 59.009 with frequency = 8
time = 59.485 with frequency = 2
time = 59.605 with frequency = 1
time = 60.558 with frequency = 5
time = 60.916 with frequency = 1
time = 61.512 with frequency = 2
time = 62.466 with frequency = 1
time = 62.585 with frequency = 1
time = 62.943 with frequency = 2
time = 63.062 with frequency = 2
time = 64.015 with frequency = 4
time = 64.135 with frequency = 1
time = 64.492 with frequency = 1
time = 64.611 with frequency = 1
time = 65.088 with frequency = 1
time = 65.565 with frequency = 2
time = 65.923 with frequency = 1
time = 66.042 with frequency = 4
time = 66.519 with frequency = 5
time = 66.996 with frequency = 8
time = 67.115 with frequency = 1
time = 67.472 with frequency = 8
time = 67.592 with frequency = 1
time = 67.949 with frequency = 1
time = 68.069 with frequency = 3
time = 68.426 with frequency = 1
time = 68.545 with frequency = 3
time = 69.022 with frequency = 4
time = 69.499 with frequency = 5
time = 69.618 with frequency = 1
time = 69.976 with frequency = 4
time = 70.095 with frequency = 3
time = 70.453 with frequency = 1
time = 70.572 with frequency = 2
time = 71.049 with frequency = 2
time = 72.122 with frequency = 1
time = 84.996 with frequency = 1
time = 236.034 with frequency = 1
time = 306.964 with frequency = 1

Appendix C: The **summary** output from a run on the lightning Opteron/InfiniPath cluster at Iowa State University with npp = 18 and the number of timing trials = 2**23.

PROGRAM PARAMETERS
Number of MPI processes used = 8
Ping pongs are between processors of rank source and dest where
source = 0
dest = 7

Number of timing trials used for parts 1, 2 & 3 are:
ntr = 16777216
npptrial = 8388608
ntrial = 8388608
Message size(# of 8-bytes els) = 1
Number of ping pongs in a timing loop (npp):
calculated value of npp = 18
value of npp used = 18

TIMING RESULTS FOR PING PONG TIMES DIVIDED BY TWO IN MICROSECONDS
minimum time = 1.2451
median time = 1.3113
average time = 1.3227
maximum time = 570.8072

STATISTICS FOR PING PONG TIMES
variance = 0.1463E+00
standard deviation = 0.3825E+00
coefficient of variation = 0.2892E+02
standard error = 0.1321E-03
relative standard error = 0.9985E-04

COMMUNICATION RATE RESULTS IN MBYTES/SECOND
communication rate using minimum time = 0.6425E+01
communication rate using median time = 0.6101E+01
communication rate using average time = 0.6048E+01
communication rate using maximum time = 0.1402E-01

FILTERED DATA RESULTS FROM PART 5
filtering cut-off coefficient = 2.0000
timing trials after filtering = 8387483
# of timings removed by filtering = 1125

FILTERED TIMING RESULTS FOR PING PONG TIMES DIVIDED BY TWO IN MICROSECONDS
minimum time = 1.2451
median time = 1.3113
average time = 1.3219
maximum time = 2.6160

STATISTICS FOR FILTERED TIMINGS
variance = 0.1207E-02
standard deviation = 0.3473E-01
coefficient of variation = 0.2628E+01
standard error = 0.1199E-04
relative standard error = 0.9073E-05

program time in minutes = 0.4683E+02