In *Computing Science and Statistics* **27** (1995), pp. 347-351.

Statistical Analysis of Communication Time on the IBM SP2

Theodore B. Tabe[1], Janis P. Hardwick[2], Quentin F. Stout[1]
University of Michigan, Ann Arbor, MI 48109

Abstract

For parallel computers, the execution time of communication routines is an important determinate of users' performance. For one parallel computer, the IBM SP2, all of the higher-level communications routines show a drop in performance as the number of processors involved in the communication increases. Such a drop is unexpected and does not occur on most other parallel machines. While a few others have also recently studied the SP2's communication performance, they have reported only average performance, and failed to comment on the drop in performance or its causes [1, 9].

We generated a distribution of times for these routines and developed a simulator in an attempt to recreate the observed distribution. By studying distributions of communication times and by refining the simulator, we were able to discern that the performance decrease is due to the variation in the communication times of the lower-level primitives upon which the higher-level communication routines are built. This variation is in turn caused by the deleterious effects of interrupts generated by an operating system untuned to high-performance parallel computing.

Keywords: all-to-all communication, resampling, performance evaluation, message passing, operating systems, interrupts, heavy tails

1 Introduction

The IBM Scalable POWERparallel Systems 9076 SP2 is a distributed memory parallel computer in which POWER2 processors are connected by a fast switch. In this distributed computing environment, processors each have their own memory space, work in parallel to compute a result, and communicate with each other by sending messages. Communication time significantly affects the performance of distributed memory systems because the cost of communicating data across the network is usually several orders of magnitude larger than the cost of computing the same amount of data. On a high performance computer, such as the SP2, code must be optimized to the highest degree if one is to obtain the maximum possible performance. Poor optimization of communication between tasks will adversely affect the scalability of a parallel application. Therefore, it is important to understand and evaluate factors affecting communication times, and to improve those that are substandard. Unfortunately, this is a very complex task.

In most parallel computer performance studies, one tends to find only mean communication times reported, with an occasional mention of corresponding standard deviations. For distributed parallel systems, however, knowing only average behavior, even along with a measure of variation, is not sufficient. The reason for this is that a typical algorithm cycles work through in stages. At each stage, a set of processors perform operations concurrently and a processor cannot send off the results of its operation until all processors have finished the stage. Thus, the measure that most affects communication times is the *maximum* operation length. Furthermore, the ultimate measures of interest here will be nested maxima since our goal is to describe higher level communication routines. For the SP2, it is especially important to consider entire distributions of events as the times of relevant communication events exhibit startlingly heavy tails.

By using the full distribution of times, we find that the drop in performance of the higher-level communication routines is due to an increase in the average processor idle time, caused by large variations in the communication times of the lower-level communication routines.

2 System Configuration

2.1 Hardware

An SP2 can be configured with from 4 to 512 POWER2 processors. The processors used in the IBM SP2 are from the IBM POWER2 family, and are the same processors found in the IBM RS/6000 series of workstations. The SP2 configuration used for this study consists of 32 Thin Node 66 processors, each of which has 256 MB of main memory and no L2 cache.

In an SP2, the POWER2 processors are connected by a High-Performance Switch (HPS). The HPS is a bidirectional

[1]Department of Electrical Engineering and Computer Science
[2]Department of Statistics

multistage interconnection network capable of 40 MB/s data transfers unidirectionally and 80 MB/s bidirectionally [6].

2.2 Operating System

Our SP2 configuration uses the AIX 3.2.5 operating system, run separately on each node within the SP2. This is the same operating system found in the RS/6000, and, as will be shown, it has not been tuned for this configuration.

The AIX operating system utilizes three different resource managers. The first is the dispatcher. It is invoked every 10 milliseconds (ms) by a timer interrupt. It is also invoked every time a process yields the CPU and after most interrupts. Therefore, the dispatcher is invoked at least 100 times a second. The dispatcher's job is to execute the process with the highest priority on the CPU [5].

The process with the highest priority is typically the scheduler. Therefore, the scheduler is usually invoked at least 100 times a second. The scheduler's job is to keep track of the CPU use of the currently running process. Additionally, once every second, the dispatcher directly executes the swapper. The swapper's job is to recalculate the priorities of all currently executing processes. The swapper also swaps processes in and out of memory [5].

2.3 Message Passing Library

Several communications packages are available on the IBM SP2, including the Message Passing Library (MPL), PVM, and MPI. MPL is the most tuned of these, and was the communication package used for this study. The MPL is a library of 32 message-passing functions that provide a programming interface through which tasks in a distributed environment may pass messages to each other. It is a proprietary system, providing functionality very similar to that of MPI.

The low-level MPL communication routine used in this study is called `mp_bsendrecv()`. This routine allows two processors to simultaneously send and receive data between them. It offers better performance than two unidirectional data transfers.

3 All-to-All Communication

The higher-level communication routine examined here is known as All-to-All communication. Other high-level routines exhibit similar phenomena, but there is not sufficient space to examine them here. A far more extensive examination of SP2 communication will appear in [7].

All-to-All communication, also known as Complete Exchange or All-to-All Personalized Communication, is defined as the communication that occurs when each processor in a group of processors wants to send a distinct message to each of the other processors in the group. It is a very important higher level communication routine, used, for example, in computing the transpose of a matrix (if each processor initially starts with a block of rows of the matrix, then to obtain a block of rows of the transpose, each processor will need to receive data from every other processor).

Several papers have been written concerning All-to-All communication, and a variety of implementations have been developed for different machines[2, 3, 4]. However, various limitations of the SP2 hardware prohibit many of these implementations (see [7]). For instance, because the bandwidth out of a processor is equal to the bandwidth of one message,

Step 1	0↔5	1↔4	2↔3
Step 2	1↔5	2↔0	3↔4
Step 3	2↔5	3↔1	4↔0
Step 4	3↔5	4↔2	0↔1
Step 5	4↔5	0↔3	1↔2

Figure 1: All-to-All Pairings for Six Processors

```
/* num_procs = number of processors */
/* my_proc = id number of processor */
/* (ids numbered 0 to num_procs-1)  */

even := (mod(num_procs,2) = 0)
if (even) then
   m := num_procs-1
else
   m := num_procs
end if

for i := 0 to m-1
    if (even .or. (i != my_proc)) then
      if (my_proc = m) then
         other_proc:=i
      else if (my_proc = i) then
         other_proc:=m
      else
         other_proc:=mod(m+i+i-my_proc,m)
      end if

      call mp_bsendrecv(other_proc)
    end if
end for
```

Figure 2: All-to-All Communication Pseudocode

Processors	All-to-All Throughput
2	40.991
4	40.928
8	40.533
16	39.436
32	38.970

Table 1: Mean Throughput in MB/s for 1MB Messages

one cannot write an All-to-All communication routine where a single processor must at some point communicate simultaneously with all the other processors in the group. Our algorithm for All-to-All communication is illustrated in Table 1, which shows the communication pattern for six processors. The pseudocode in Figure 2 shows in detail how the All-to-All communication algorithm works. Based on timing comparisons, we believe that the MPL implementation is essentially this algorithm, and from now on all references to All-to-All communication will assume that this algorithm is being used. (Due to space limitations, we omit the timing studies comparing the two versions.) The specific exchange pattern of this All-to-All communication algorithm was obtained from Andrew Poe, who encountered it as a way to pair up players for chess tournaments.

Table 1 illustrates the basic problem that arises with All-to-All communication on the IBM SP2, namely,

> For a fixed message length, the throughput of the communication routine drops as the number of processors is increased.

Here *throughput* is the number of megabytes per second transmitted and received by a given processor. In principle, such a decrease should not occur because, for any number of processors, each step of the algorithm should take the same amount of time.

4 One-to-One Communication

To understand the phenomena for the higher-level All-to-All communication, we examined the lower-level communication routine of which it was comprised. This is the One-to-One communication or ‘pairwise exchange’ routine `mp_bsendrecv()`. Figure 3 shows the distribution of times exhibited by this routine. The mean is 173 microseconds, and the standard deviation is 58 microseconds. For parallel communication, this represents a very high coefficient of variation of 0.34. On the nCUBE1, for example, the mean for the same routine was 4.5 milliseconds and the standard deviation was less than 0.05 milliseconds [8], giving a coefficient of variation of 0.01. Moreover, on the SP2 the distribution of times

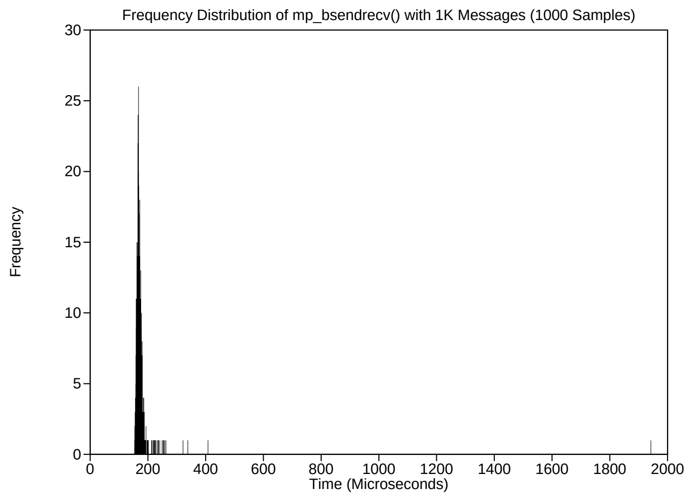

Figure 3: Frequency Distribution of Point-to-Point Exchange for 1024 Byte Messages

is highly skewed to the right.

The large variation in times observed is caused by the fact that the communication routines can be delayed by operating system interrupts since they are not kernel routines. Figure 4 shows the occurrences of operating system interrupts on an SP2 node over time. The bottom thick, black line represents time measurements that were not interrupted. The row of points above this are the dispatcher interrupts. The row of points above the dispatcher interrupts is another set of timer interrupts of unknown origin. The time between these interrupts has a mean of 6025 microseconds. Finally, the very large times (greater than 200 microseconds) are probably page faults of the operating system.

If one simulates one-to-one communication by resampling from the uninterrupted communication times along with the interrupts, one can reproduce the empirical distribution of one-to-one communication. This simulation is discussed in [7].

5 All-to-All Simulator

To understand why variations in point-to-point communications cause performance degradation, consider the pairwise exchanges in Table 1. For the final 4↔5 exchange to occur in Step 5, the 3↔5 and 4↔2 exchanges of Step 4 must both be finished, which means that the 4↔5 exchange must wait for the slower of the two predecessors, not their average. Further, each exchange in Step 4 must wait for two exchanges at Step 3, and so on. If any one of these encounters an operating system page fault, for example, then all of the subsequent exchanges will be delayed. The number of predecessors of a final exchange grows quadratically in the number of proces-

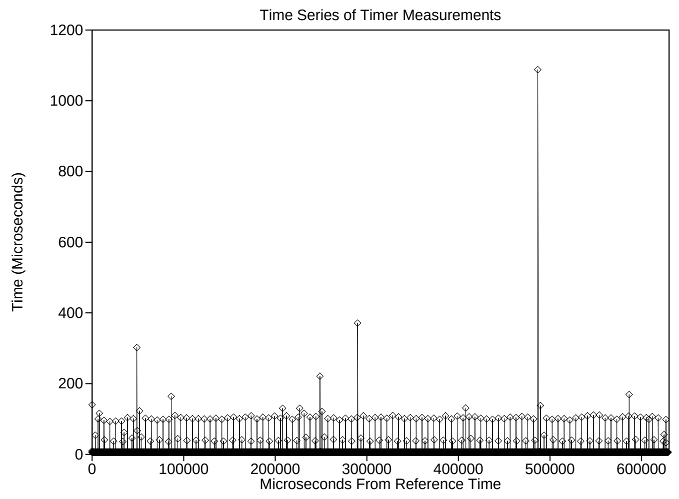

Figure 4: Time Series of Operating System Interrupts on an SP2 Node

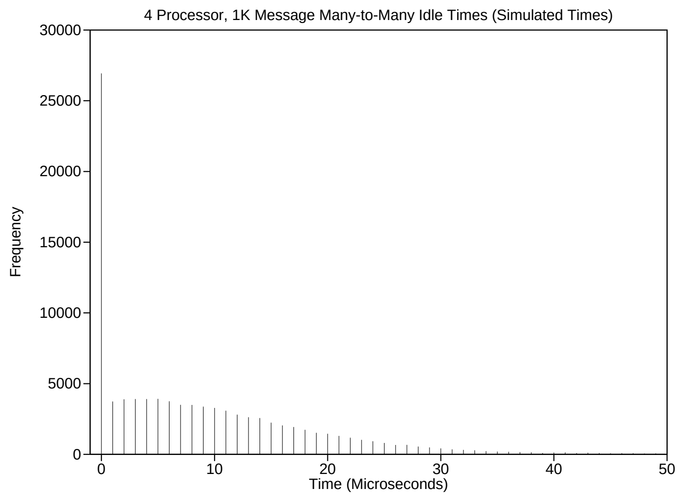

Figure 5: Idle Time for Processor 0, All-to-All Communication, 4 Processors

sors, and the number of dependence paths grows exponentially. It is the slowest of these paths that corresponds to the total time of a All-to-All communication, and thus, it is the tail behavior of the individual components (the pairwise exchanges) that becomes critical.

To test our hypothesis that it was the variation of the pairwise exchanges that caused the degradation of all-to-all communication, we constructed a simple simulator. By gathering a large sample of timings for pairwise exchanges, we hoped to to create a fairly accurate distribution of the All-to-All communication times. Each simulated All-to-All sample is generated by assigning a random communication time from the sample set to each pairwise exchange of data. With this set of pairwise exchange times, one can calculate the total communication time for an arbitrary processor and the total idle time for each processor. (Processor p is *idle* if it is ready to begin communication with processor q, but q is not yet ready and hence p must wait. Typically q is not yet ready because it has not finished its previous communication with a third processor.) The simulator was written in C and the random number generator used was the `random()` library call.

The timings from the simulator exhibited a distribution similar to that of the times measured on the SP2 except that the means did not coincide. It is likely that this is because our simulator is insufficiently refined. The initial simulator did not account for the fact that the operating system interrupts on a processor are correlated. If a given processor has not had a system interrupt recently, then the probability that it will be interrupted in the near future increases. We are incorporating this effect in the simulator that we are presently developing, using interrupt timing traces.

Peak All-to-All communication performance is achieved by having all processors have total idle times of 0. Due to operating system interrupts and page faults, there is variation in the processor-to-processor communication time and, therefore, the idle time is much greater than 0. Figure 5 shows the idle time measured in the simulator for the 4 processor case, and Figure 6 shows the simulator results for the 32 processor case. In all simulator cases, 100,000 sample communications were created. As one can see, the number of communications where the idle time is 0 drops drastically as the number of processors increases. Also, the mean and the maximum size of the idle times increases with the number of processors. This increase in processor idle time explains the drop in throughput of the All-to-All Communication.

6 Conclusions

There are a number of conclusions that can reached from this research. First, it must be emphasized that using only means to model communications performance of parallel computers is inadequate. As popular as they are in the performance literature, central tendency measures lack the information available in full distributions. Understanding the tail behavior of relevant distributions is critical to the development of good simulators.

Next, we have seen that, to maximize performance of the All-to-All communication routine, one must minimize the variation in the processor-to-processor communication time. Furthermore, it is important to understand the behavior of the lower-level communication primitives upon which a higher-level communication is based in order to gain insight into the behavior of the higher-level communication primitives.

Finally, we believe that the drop in performance of the All-to-All communication routine is due to the unexpected effects

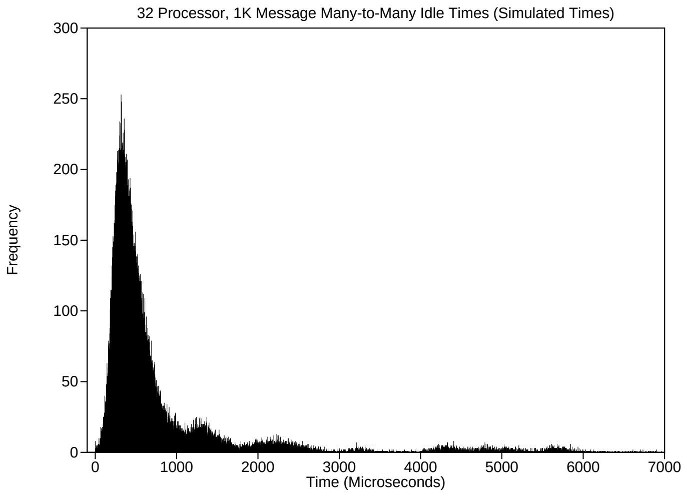

Figure 6: Idle Time for Processor 0, All-to-All Communication, 32 Processors

of creating a parallel machine from general-purpose processors running a general-purpose operating system. The AIX operating system was not designed for a high-performance, single-user parallel processing environment. It has many more interrupts than are normal for such machines, and their asynchronous occurrence magnifies the problem. As more and more parallel machines are built in this manner, we shall probably see more unexpected phenomena like this where a drop in performance is caused by the fact that some part of the parallel computer was not designed to be run in a parallel environment.

Acknowledgments

Craig B. Stunkel at IBM Thomas J. Watson Research Center provided a great deal of help in understanding the details of the inner workings of the SP2 communication layer.

Computing services were provided by the University of Michigan Center for Parallel Computing, which is partially funded by NSF grant CDA-92- 14296.

This research is partially supported by NSF grant DMS-9157715.

References

[1] E. L. Boyd, G. A. Abandah, H.-H. Lee, and E. S. Davidson, “Modeling Computation and Communications Performance of Parallel Scientific Applications: A Case Study of the IBM SP2”, draft submitted to *Supercomputing ’95*.

[2] S. Hinrichs, C. Kosak, D. R. O’Hallaron, T. M. Stricker, and R. Take, “An Architecture for Optimal All-to-All Personalized Communication”, Technical Report CMU-CS-94-140, Carnegie Mellon University, September 1994.

[3] S. L. Johnsson, C.-T. Ho, “Optimal All-to-All Personalized Communication with Minimum Span on Boolean Cubes”,Technical Report TR-18-91, Harvard University, April 1991.

[4] K. K. Mathur, S. L. Johnsson, “All-to-All Communication on the Connection Machine CM-200”, Technical Report TR-02-93, Havard University, January 1993.

[5] R. Mraz, “Reducing the Variance of Point-to-Point Transfers for Parallel Real-Time Programs”, http://ibm.tc.cornell.edu/ibm/pps/doc.

[6] C. B. Stunkel et al., “The SP2 Communication Subsystem”, http://ibm.tc.cornell.edu/ibm/pps/doc/.

[7] T. Tabe, J. Hardwick, and Q.F. Stout, “Performance Analysis of Communication on the IBM SP2”, in preparation.

[8] B. A. Wagar, “Practical Sorting Algorithms for Hypercube Computers”, PhD Thesis, University of Michigan, 1990.

[9] Zhiwei Xu, “Is SP2 the best supercomputer?”, posting to USENET group *comp.parallel*, University of Southern California, April 16, 1995.