NASA Contractor Report 198211

ICASE Interim Report No. 28

ICASE

COMMUNICATION OVERHEAD ON THE INTEL PARAGON, IBM SP2 & MEIKO CS-2

Shahid H. Bokhari

NASA Contract No. NAS1-19480
September 1995

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23681-0001

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

Communication Overhead on the Intel Paragon, IBM SP2 & Meiko CS-2

Shahid H. Bokhari
Department of Electrical Engineering
University of Engineering & Technology, Lahore, Pakistan

Abstract

Interprocessor communication overhead is a crucial measure of the power of parallel computing systems—its impact can severely limit the performance of parallel programs. This report presents measurements of communication overhead on three contemporary commercial multicomputer systems: the Intel Paragon, the IBM SP2 and the Meiko CS-2. In each case the time to communicate between processors is presented as a function of message length. The time for global synchronization and memory access is discussed. The performance of these machines in emulating hypercubes and executing random pairwise exchanges is also investigated.

It is shown that the interprocessor communication time depends heavily on the specific communication pattern required. These observations contradict the commonly held belief that communication overhead on contemporary machines is independent of the placement of tasks on processors. The information presented in this report permits the evaluation of the efficiency of parallel algorithm implementations against standard baselines.

Research supported by the National Aeronautics and Space Administration under NASA contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science & Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-0001.

Work on the Intel Paragon was performed using the CACR parallel computer system operated by Caltech on behalf of the Center for Advanced Computing Research. Access to this facility was provided by NASA.

Work on the IBM-SP2 was performed using the resources of the Cornell Theory Center, which receives major funding from the National Science Foundation and New York State with additional support from the Advanced Research Projects Agency, the National Center for Research Resources at the National Institutes of Health, IBM Corporation, and members of the Corporate Research Institute.

Work on the Meiko CS-2 was carried out using the resources of the Vienna Center for Parallel Computing, funded as part of the European ESPRIT project PPPE, and was supported by the Institute for Software Technology and Parallel Systems of the University of Vienna.

1 Introduction

Interprocessor communication is a key issue in parallel computing. The impact of communication overhead can severely limit the performance of parallel algorithms on massively parallel processors. Considerable effort is required to minimize the overhead of interprocessor communications. In this report we investigate the communication performance of three contemporary commercial multicomputer systems: the Intel Paragon, IBM SP2 and Meiko CS-2. All three machines incorporate powerful interprocessor communication mechanisms. They each have special purpose hardware dedicated to interprocessor communications as well as very powerful computational nodes. The Paragon has a mesh interconnect while the SP2 and CS-2 use multistage networks.

We investigate the time required to communicate between processors, the time to execute barrier synchronization, and the time to move bytes within a processor's memory. These are the central operations required in interprocessor communications. We also explore how these machines behave when they are made to emulate hypercubes and when they execute random pairwise exchanges. Contrary to the commonly held belief that in modern multicomputers the physical locations of the tasks comprising a parallel computation are unimportant for performance, we demonstrate that nearly 100% degradation can be experienced when tasks are poorly mapped.

On the Intel Paragon, whose mesh architecture permits us easily to contrive very stressful communication patterns, we can show a degradation of a factor of 3 due to link contention. On the SP2, contention for switches because of jobs other than the test job can cause very large variations in communication time that make it difficult for the programmer to gauge the efficiency of an algorithm's implementation.

We conclude this report by commenting on the difficulties encountered in programming these systems. These comments are designed to highlight areas where improvements could be made by vendors to make the onerous task of parallel programming less burdensome for the user.

1.1 The Intel Paragon

The Intel Paragon[1] on which the experiments described in this report were carried out is located at the Center for Advanced Computing Research at Caltech[2]. It is made up of 512 compute nodes organized in a 16×32 array. Each node is composed of two Intel i860 processors. One serves as a compute processor and the other as a communication processor. In addition there is special hardware for interfacing with the intercommunication network. The interprocessor communication network is a mesh with "row–column" routing. That is, a message from processor x_1, y_1 to processor x_2, y_2 first travels along a row to x_2, y_1 and then along a column to x_2, y_2. This routing algorithm is fixed and cannot be modified by the user. A message passing through a node en route to its destination does not impact the computation occuring at that node as the routing is carried out by special routing hardware. The i860s run at 50 MHz and are capable of 75 MFlops.

This machine has 32 Megabytes of memory per node of which about 24 Megabytes are available for user programs. It also supports virtual memory, which complicates measurements of communication overhead, since the overhead of paging in data through the communication network interferes with the experiments being conducted on the network. For this reason it is important to run all programs with the `-plk` (process lock) option, which locks all pages in memory. Even with this option there is some paging when a program is started. The perturbation caused by this can only be circumvented by performing an experiment twice and discarding the timings for the first run.

Each node of the Paragon runs the OSF/1 operating system. This is a full fledged Unix-like operating system in which there are many system activities going on at the same time that a user program is running. In particular there are four types of interrupts that occur periodically for various operating system functions. These complicate the measurement and control of communication on this machine. Shirley et al. [7] discuss this issue in detail.

The `nx` message passing library was used for the experiments described in this report. This library has its origins in the iPSC-860 hypercube which has two types of messages: `FORCED` and `UNFORCED`. `FORCED` messages are trans-

[1]`http://www.ssd.intel.com/paragon.html`
[2]`http://www.ccsf.caltech.edu`

mitted from source to destination under the assumption that a receive has already been posted (i.e. buffer space for reception specified) at the destination. If an arriving message does not find a receive posted, it is discarded. `UNFORCED` messages do not require a receive to be posted beforehand. Before an `UNFORCED` message is transmitted there is an exchange of control messages between source and destination to allocate operating system buffer space for the message. This leads to additional overhead in communication (because of the control messages), extra memory requirements, and the penalty of copying from operating system buffers to user areas [1].

On the Paragon, `FORCED` and `UNFORCED` messages are *supposed* to perform identically. It has been our experience that operating system space is allocated for all possible arriving messages *in addition* to any user memory locations that may be set aside by explicitly posted receives. The user can specify the amount of memory buffers that the operating system is to set aside for this purpose using the `-mbf` run option. Despite this, when large numbers of large-sized messages are expected, the operating system can run out of resources thereby causing the machine to hang (even if adequate memory has been set aside with the `-mbf` option.) In our experience, a 16×16 submesh of the Caltech Paragon cannot handle 255 receives of 1792 or more bytes each.

Needless to say, `FORCED` messages should only be used if the communication requirements are well understood and receives can be posted before any messages are launched. Deadlocks can develop if this requirement is not satisfied.

Disk I/O on the Paragon tales place through the same network that is used for interprocessor communication. All disks are attached to separate I/O processors on the left and right-hand boundaries of the machine. Should user A's jobs be running on processors located between user B's job and B's disk(s), A will suffer considerable overhead. This interference is easy to spot in a series of timing measurements as it shows up as a sharp spike on an otherwise smooth plot. It can be avoided by placing a job judiciously on the mesh or running in dedicated mode. A very useful utility called `xpartinfo` provides a visual representation of the jobs on the machine for this purpose.

Memory access and thus data communication on the Paragon is heavily affected by the starting address of a transfer. In our experiments we have aligned all arrays to 4k boundaries (the page size of the machine) to minimize this impact.

1.2 The IBM SP2

The Cornell Theory Center[3] has a 512 node IBM SP2[4] multicomputer that was used for these experiments. Each node on this machine is made up of POWER2 architecture RS/6000 processors running at 66.7 MHZ with at least 128 MBytes of memory each. All nodes run AIX, which is IBM's version of Unix. Each machine is theoretically capable of 266 MFlops. The machines are interconnected with a multistage switch made up of bidirectional 4×4 crossbars. Special hardware interfaces the compute processors with the interconnection network [3].

The `MPI-F` message passing library [5], a version of `MPI` [4, 6] for the IBM SP2, was used to implement the programs. SP2 interprocessor communication can be subdivided into two types, roughly analogous to `FORCED` and `UNFORCED` messages on the Intel iPSC860, which were discussed in Section 1.1. Messages shorter than a predefined limit m are sent directly to their destinations. Messages larger than this require an exchange of control messages prior to actual transmission. The control messages can have significant overhead, especially for small message sizes. The default value of m is 4096 bytes. This can be overridden using the `-mpiS`m runtime flag which sets to m the threshold at which control messages start being used. m is limited to 16384.

The `MPI_wclock()` routine that is supplied with `MPI-F` has poor resolution. We found that the Fortran real time clock `rtc()` routine could be linked in and provided satisfactory resolution.

When a large parallel application is run on the SP2, the processors allocated to the application are scattered all over the machine. There may be contention at the switches caused by other jobs. The impact is quite significant and is unavoidable as there is no way for the user to control where a job is placed. Operating system events on the processors themselves also cause some perturbation.

[3]`http://www.tc.cornell.edu`

[4]`http://ibm.tc.cornell.edu/ibm/pps/sp2/`

1.3 The Meiko CS-2

The Meiko CS-2[5] at the Vienna Center for Parallel Computing[6] is made up of 128 SuperSPARC processors running at 50 MHz. Each processor has 64 MBytes of memory, runs the Solaris operating system, and is capable of 100MFlops. Special purpose hardware connects each processor to a multistage switch which is made up of bidirectional 4×4 crossbars.

Several programming systems are available on the CS-2. We have used the `mpsc` library which permits `nx` programs written for the Intel hypercubes, Touchstone-Delta or Paragon to be executed virtually without change. Like the SP2 `MPI-F`, the CS-2 `mpsc` library suffers from a poor resolution clock routine. We were, however, able to access the high resolution `elan` clock from within our programs. The CS-2 is the newest of the 3 machines tested and was thus lightly loaded. It was easy to avoid the impact of other users' communications on our experiments. We anticipate that, as the use of this machine increases, this impact will become more noticeable.

The specific CS-2 at Vienna does not have a full complement of switches at the higher levels of its multistage connect. As a result it suffers from discontinuities in performance as we move from 32 to 64 processors. Because of hardware problems, we were unable to experiment with more than 64 nodes on this machine. Of the 3 machines evaluated in this report, the CS-2 has the distinction of having the most linear communication times and the best barrier synchronization time.

2 Communication Overhead

Communication overhead was measured on all three machines for messages varying in length from 0 to 16000 bytes in steps of 64 bytes. The general strategy for these experiments is to exchange messages between two nodes `ITERS` times and then divide the total elapsed time by $2\times$`ITERS`. It is necessary to have `ITERS` set to a value large enough to reduce the impact of the perturbation caused by the overhead of clock reading as well as to reduce the statistical variance caused by interference from messages due to other users or from operating system events.

[5] `http://www.meiko.com/`

[6] `http://www.vcpc.univie.ac.at/vcpc.html`

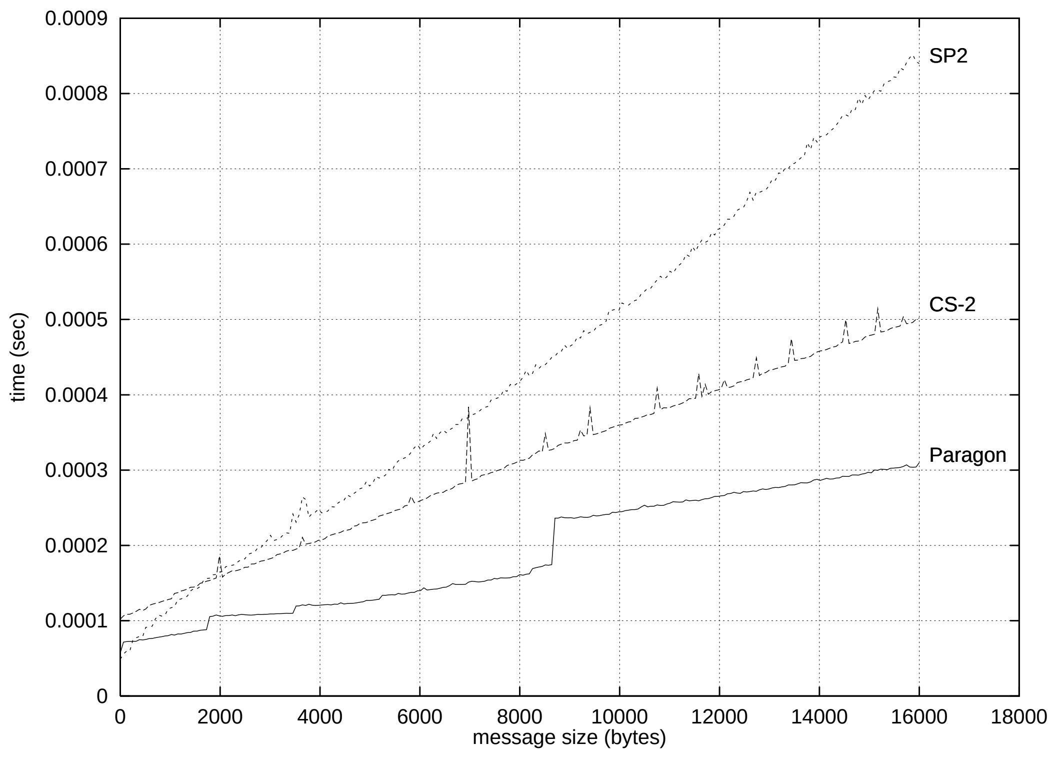

Figure 1: Communication times on the Paragon, SP2 and CS-2.

Each of the two processors involved was made to post `ITERS` receives, which was followed by a barrier synchronization. The two processors then exchanged `ITERS` messages and then synchronized again. The total time, which included the time to execute two barriers, was divided by $2 \times$`ITERS`. The contribution to the total time by the barriers is negligible because `ITERS` ≥ 200 in all cases. On the Paragon and CS-2 we used `ITERS` $= 200$. On the SP2 it was necessary to go to `ITERS` $= 1000$ to reduce the impact of fluctuations caused by other jobs.

Non blocking sends and receives were used so as to exploit the parallelism that is possible in machines that have a separate processor for communication. The use of blocking sends is specifically recommended in the documentation of the SP2. However we found no difference in blocking and non blocking sends on that machine.

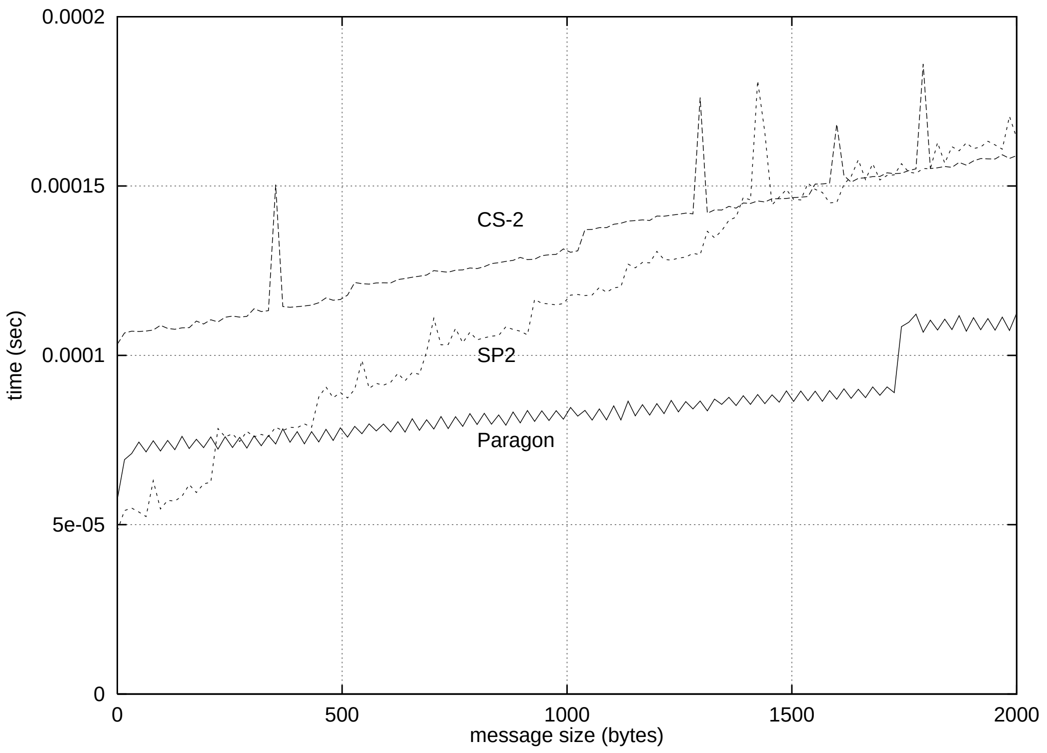

Figure 2: Communication times for messages smaller than 2000 bytes.

Figure 1 shows the run times for the three machines for message sizes varying from 0 to 16000 bytes in increments of 64 bytes. Figure 2 shows a more detailed plot for message sizes from 0 to 2000 bytes in increments of 16 bytes. The steps in the Paragon plots are caused by packetization overhead. On the SP2 and CS-2 plots there are spikes on the plots caused by interference from other users' messages or from operating system events. Expressions for communication time are given in Table 1. For each machine we give the time for zero byte, short and long messages. The performance of all three machines for zero byte messages is significantly better than the performance predicted by the short message expressions. We have included this information because zero byte messages can be employed to send useful information (through message type tags). The dividing point between short and long messages is 8640 bytes for the Paragon and 10000 bytes for the SP2.

	Mem. copy	Barrier (μsec)	Communication (μsec per byte)		
	(μsec/byte)	2^d procs.	zero	small	large
Paragon	0.0140	$126d - 113$	58	$75 + 0.011m$	$135 + 0.011m$
SP2	0.0043	$72d - 52$	48	$70 + 0.043m$	$-50 + 0.056m$
CS-2	0.0153	$17d - 5.6$	103	$105 + 0.025m$	$105 + 0.025m$

Table 1: Summary of performance figures

In Table 1 we have supplied linear curve fits from Figure 1 for these ranges. The communication times for the SP2 have a distinct upward curve that we have approximated with two straight line fits. The expression $-50 + 0.056m$ only applies to large (> 10000 byte) messages.

3 Synchronization Time

Barrier synchronization is an important operation in parallel programming. The time required for this operation was measured on the Paragon and CS-2 using a simple loop that executed `gsync()` one hundred times. On the Paragon, the measurements were for square meshes of size 2×2, 2×4, 4×4, $\ldots$, 8×16, 16×16. On the SP2 and CS-2 we measured on processor pools of $2, 4, \ldots, 64$ processors.

On the SP2 there is a great deal of spread in the timings for `MPI_Barrier` due to interference from other messages. For this machine scatter plots were obtained and a line fitted through the minimum time for each machine size. Although this gives the benefit of the doubt to the machine and leads to a clean expression, it is not an accurate prediction of the synchronization time that will be encountered by the average user.

For all three machines the time for synchronization is logarithmic in the number of processors and is summarized in Table 1.

4 Memory Access Time

Many communication operations are organized as sequences of message transmissions alternating with data permutation. The multiphase complete exchange is one example [2]. For such communication operations the time required to move blocks of data from one part of memory to another *on the same processor* can heavily influence the total time required.

The best way to move large blocks of data in memory is to use the C `memcpy()` routine. Using this routine we have measured the memory to memory transfer times (in nanoseconds) on the three machines to be 14 for the Paragon, 15.3 for the CS-2, and 4.3 for the SP2. The timings on the Paragon are particularly sensitive to the starting address of the transfer: the 14 nanosec. figure is for the case where the starting address is a multiple of 16. Failing this, the time per byte can go up by an order of magnitude. The CS-2 and SP2 do not suffer from such problems.

5 Emulating the hypercube

Much of the early parallel algorithm research work was targeted for hypercubes. None of the three machines being considered in this report is a hypercube, although they do have high bandwidth interconnects. It is interesting to investigate how well these machines do when *emulating* hypercubes. To this end we wrote an emulation program to make each machine assume that it was a hypercube of dimension d and then successively communicate over all d dimensions[7]. Since these machines do not, of course, include the hypercube interconnect as a subset of their connections, we would expect to see poorer performance than if we were communicating between pairs of isolated processors. However we argue that if there is a linear increase in time as the dimension of the emulated hypercube increases, we can conclude that the emulation is a success, since the time to communicate over all dimensions in a true hypercube is linear in the number of dimensions.

The results of this experiment are given in Figure 3. It can be seen that the Paragon and SP2 satisfy the linearity requirement reasonably well. On the Paragon there is a discontinuity in most plots at slightly over 8000 bytes, but this mirrors the discontinuity in simple communication timings (Figure 1) and should be expected. On the SP2 the plots for dimension 6 and 7 appear to be corrupted by contention at the switch, but the underlying plots are nevertheless uniformly spaced.

On the CS-2 we have uniformly spaced plots for $d = 1, \ldots, 4$ but there are discontinuities going from 4 to 5 and then from 5 to 6. This is because

[7]for $i = 0$ to $d-1$, j communicates with $j \oplus 2^i$, $0 \leq j < d$

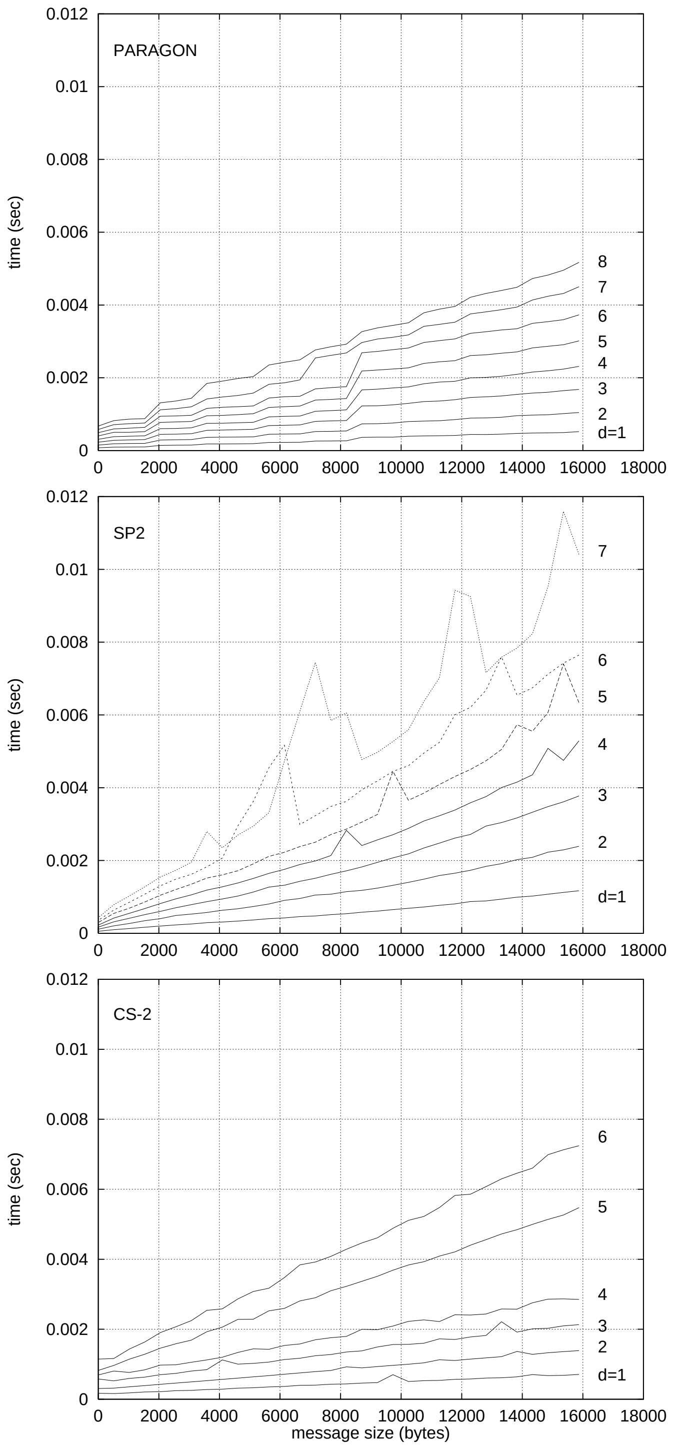

Figure 3: Hypercube emulation.

the Vienna CS-2 does not have a full complement of switches at the higher levels of the multistage network, leading to contention for switches and communication links.

These results, although welcome, should be interpreted with great care. We have only shown how these machines perform when communicating across dimensions. A real hypercube with circuit switched or wormhole routed communications is capable of very powerful communication steps across several dimensions at a time. The results of our experiment do not *necessarily* imply that the Paragon, SP2 and CS-2 can emulate the hypercube in this respect.

When carrying out this experiment on the Paragon, no attempt was made to map hypercube nodes onto mesh nodes so as to minimize edge contention, as suggested for the iWarp by Stricker [8]. We do not know of corresponding mappings for the SP2 and CS-2 and thus the Paragon would have had an unfair advantage. However, we can see in Figure 3 that the Paragon is doing quite well, even without an intelligent mapping scheme.

6 Random Permutations

It is often argued that the high speed interconnect of contemporary multicomputers, coupled with their use of dedicated communication hardware, frees the programmer from having to be concerned about the mapping of tasks to processors. According to this line of reasoning the programmer only has to be concerned about local and non-local accesses, since the high performance interconnect makes communication between a given processor and all remaining processors equal in cost.

To illustrate the falsity of this argument we measured the time required to communicate over the edges of random matchings of 64 nodes. This experiment was carried out on 8×8 submeshes on the Paragon and 64 node processor pools on the SP2 and CS-2. 32 pairs of nodes were randomly selected and made to exchange messages (200 each for the Paragon and CS-2, 1000 for the SP-2). For each permutation the message size was varied from 0 to 16384 bytes in steps of 1024 bytes. This process was repeated for a total of 256 random matchings.

If the time required to communicate were indeed independent of the mapping, we would expect to see the run times closely clustered together. In actual fact Figure 4 shows there to be very wide variation. For the Paragon

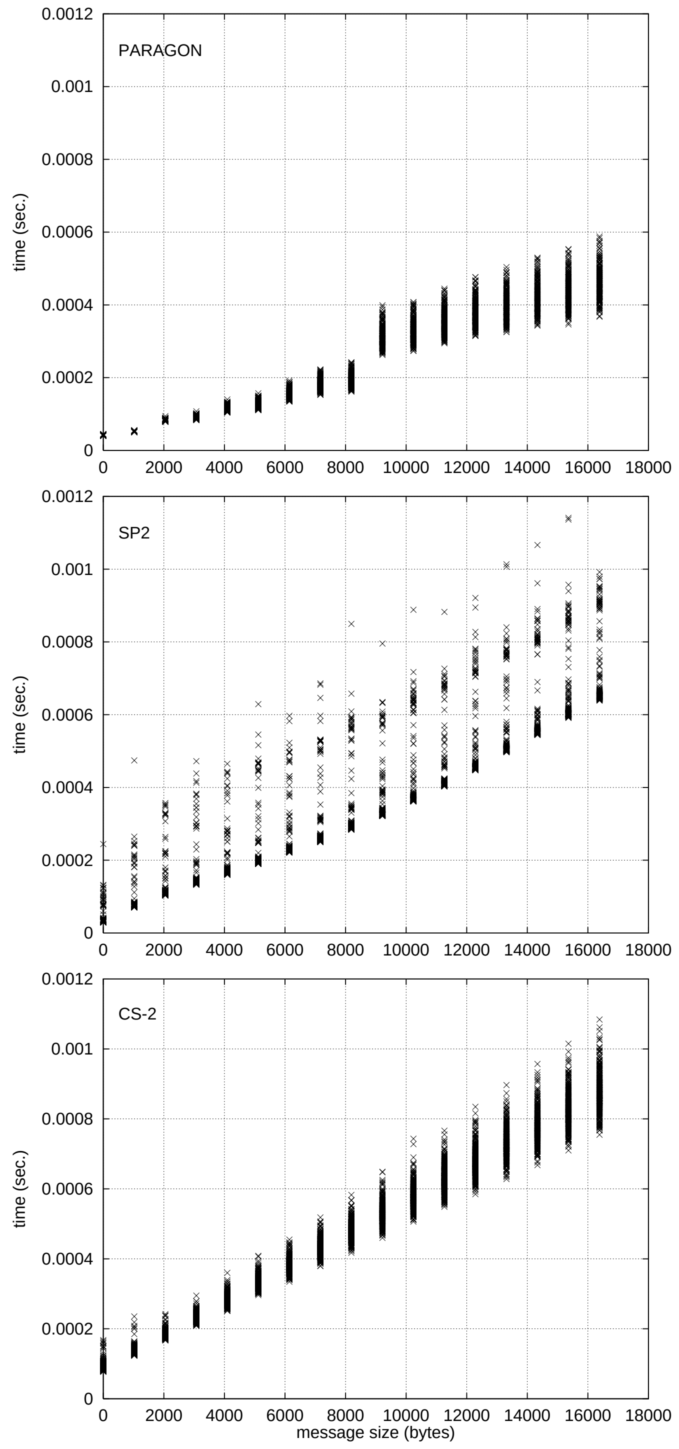

Figure 4: 256 random matchings mapped on 64 processors.

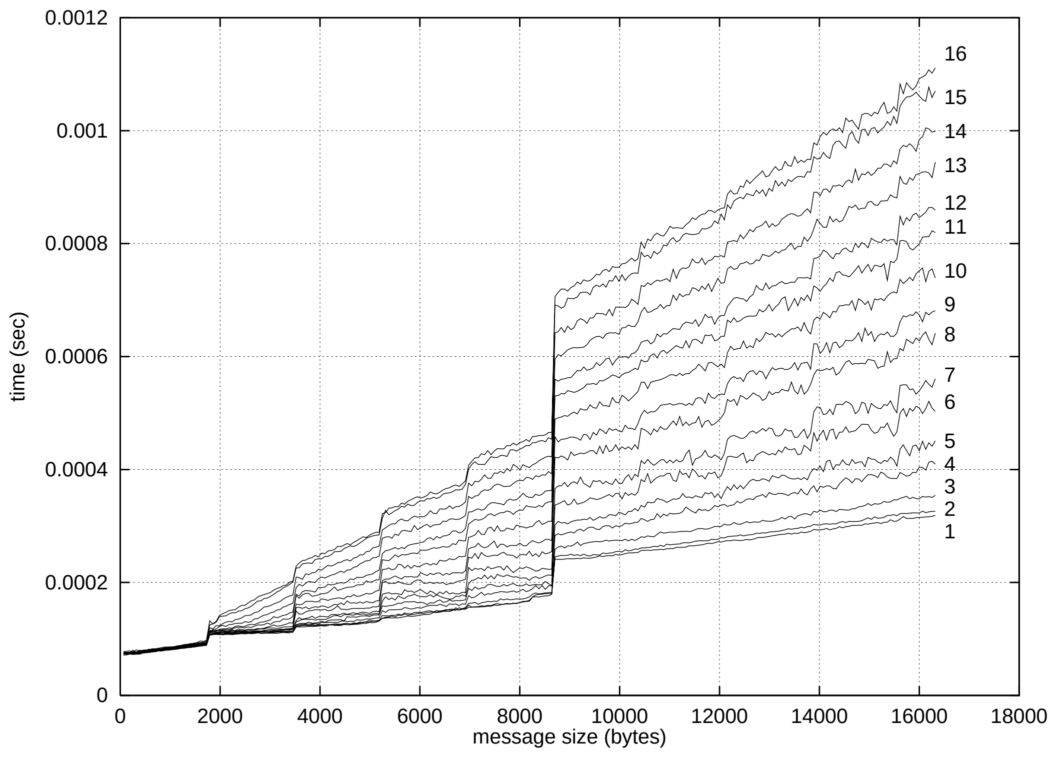

Figure 5: Contention on the Paragon.

the variation is about 60% and for the CS-2 about 25%. On the SP-2 the variation can exceed 100%. In this figure the individual plots for each of the 256 mappings are not monotonic. It would therefore be meaningless to join the points with lines.

This experiment illustrates the impact of link and switch contention when executing matchings, which are the simplest permutations possible. In actual practice much more complex permutations need to be executed and the spread of times is likely to be even greater.

7 Contention Overhead on the Paragon

The Paragon has a mesh interconnect and a simple row–column routing strategy. On this machine it is easy to contrive communication patterns which give rise to link contention, i.e., where a communication link is required for the transmission of 2, 3, 4 and more messages. A simple way of doing this is to select a chain of size n, with processors numbered $0, 1, \ldots, n-1$, and to make processor i exchange messages with processor $n-i$, for $0 \leq i < n/2$. In this experiment the maximum contention (in the link joining processors $n/2-1$ and $n/2$) is $n/2$. For the case $n = 2$ only one pair of processors contends for a link and the contention is 1. For $n = 32$ processors (the longest chain possible in the Caltech Paragon) the contention is 16. Figure 5 shows the result of this experiment. As the contention level increases, the time required goes up steadily. At maximum contention, the time to communicate increases by a factor of almost 3. The "staircase" in these plots is due to packetization of messages and each step occurs at the packet size of 1792 bytes.

At this point in time the routing algorithms for the SP2 and CS-2 are not well known, nor is there any control over task placement. Thus we are unable to contrive similar extreme contention experiments. However the results of Section 6 show that the SP2 and CS-2 also exhibit considerable sensitivity to communication patterns.

8 Discussion

We have presented the results of several experiments that measure key aspects of interprocessor communications on the Intel Paragon, IBM SP2 and Meiko CS-2. Parallel computers such as these continually evolve as operating systems mature and hardware is upgraded. Thus the specific figures presented in this report may soon be superseded. However this research does bring out several important problems that are not commonly acknowledged.

Firstly, the overhead of interprocessor communication is substantial and very sensitive to the placement of tasks on processors. Secondly, the effects of contention for communication links and/or switches can be quite severe. This contention may be caused by the messages of the job itself as well as by interference from other users' messages. These observations contradict commonly held beliefs popularized by vendors and parallel computing enthusiasts.

Among other issues that were faced during the course of this work are the following.

1. The IBM SP2 (under `MPI-F`) and the the Meiko CS-2 (under `mpsc`)

lack access to a high resolution clock. The user has to go through a convoluted effort to find out about this and then link in a clock from another package. Given that millions of dollars are spent on each of these machines, and given that the purpose of supercomputers is to deliver high performance, it is inexcusable for manufacturers to omit so essential a feature as access to a high resolution clock.

2. On machines that use paging and caching, performance is heavily influenced by the overhead of paging in data blocks and by the specific memory addresses where data transfers originate. These idiosyncrasies need to be more widely known to programmers who need better tools to help gauge their effects.

3. All three machines evaluated run full fledged operating systems on their compute nodes. Timer interrupts due to operating systems activity may disrupt data transfers and make the task of predicting performance very difficult.

4. Vendor engineers and programmers often lack understanding of basic concepts of interprocessor communications. When hardware or software problems were uncovered, the author had to spend a great deal of his time explaining why he was carrying out these experiments.

5. The Paragon and SP2 are run in "glass–house" queuing environments reminiscent of the 1960's. In some cases this isolates the user from the machine to the extent that he cannot react rapidly to the output of an experiment. The Caltech Paragon splits the day into interactive time and queuing time and alleviates this problem to a great extent.

6. Visualizing the load on a system is important for the experimenter. The `xpartinfo` utility on the Caltech Paragon permits a user to see who is running where. There is no comparable facility on the Cornell SP2: a request for queue information returns several pagefuls of data that are difficult to interpret.

The above critique is designed to highlight areas where improvements can be made by vendors and computer center administrators. Such improvements will boost the development of parallel computing by making the onerous task of parallel programming less burdensome for the user.

Acknowledgments

I am grateful to Yousuff Hussaini at ICASE for his encouragement of this research. I have received valuable assistance and advice from Tom Crockett, David Keyes, Piyush Mehrotra, David Nicol and John van Rosendale. Discussions with Steve Seidel, Paul Fischer and Xian-He Sun have also been very helpful.

I wish to thank Hans Zima and Barbara Chapman of the University of Vienna for hosting my stay in Vienna and for the use of the Meiko CS-2. Vestica Aleksandar, Thomas Fahringer, Ian Glendenning, and Hans Mortisch provided generous assistance in Vienna. James Cownie of Meiko was very helpful in providing information on the high resolution clock on the CS-2.

Access to the supercomputers at Caltech was arranged by Paul Messina. I wish to thank him and his able staff: Walker Aumann, Clark Chang, Shay Chinn, Alex Leung, Jan Lindheim, Heidi Lorenz-Wirzba, Julie Murphy, Mark L. Neidengard, Gary Dell'Osso, and Elsa Villate, for their alacrity in answering my queries and patiently tolerating the numerous crashes I caused on their machines. Thanh Phung, Al Bessey and Ellen Deleganes of Intel helped me with various problems on the Paragon.

Peter M. Siegel kindly allocated an account on the IBM SP2 at Cornell. Pat Colasurdo, Cindy Harwzinske, Marcia Pottle, Nikki Reynolds, Rachel Smith, Daniel Sverdlik and Carol Webster were very helpful during my work on the SP2.

References

[1] S. H. Bokhari. Communication overheads on the Intel iPSC-860 hypercube. ICASE Interim Report 10, May 1990.

[2] S. H. Bokhari. Multiphase complete exchange on Paragon, SP2 and CS-2. Technical Report 95-61, ICASE, September 1995.

[3] Craig Stunkel *et al.* The SP2 communication subsystem. Technical report, IBM T. J. Watson Research Center, August 22, 1994.

[4] Message Passing Interface Forum. MPI: A message passing interface standard. Technical report, University of Tennessee, Knoxville, June 12, 1995.

[5] Hubertus Franke. MPI-F: An MPI implementation for IBM SP-1/SP-2. Technical report, IBM T. J. Watson Research Center, May 1995.

[6] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message–Passing Interface.* MIT Press, Cambridge, Massachusetts, 1994.

[7] Hazel Shirley, Robert Reynolds, and Steve R. Seidel. Communication on the Intel Paragon. Technical Report CS-TR-95-07, Dept. of Computer Science, Michigan Tech. Univ., July 17, 1995.

[8] Thomas M. Stricker. Supporting the hypercube programming model on mesh architectures. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, 1992.