Multiphase Complete Exchange on Paragon, SP2 & CS-2

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

Abstract

The overhead of interprocessor communication is a major factor in limiting the performance of parallel computer systems. The *complete exchange* is the severest communication pattern in that it requires each processor to send a distinct message to every other processor. This pattern is at the heart of many important parallel applications. On hypercubes, *multiphase* complete exchange has been developed and shown to provide optimal performance over varying message sizes.

Most commercial multicomputer systems do not have a hypercube interconnect. However they use special purpose hardware and dedicated communication processors to achieve very high performance communication and can be made to *emulate* the hypercube quite well.

Multiphase complete exchange has been implemented on three contemporary parallel architectures: the Intel Paragon, IBM SP2 and Meiko CS-2. The essential features of these machines are described and their basic interprocessor communication overheads are discussed. The performance of multiphase complete exchange is evaluated on each machine. It is shown that the theoretical ideas developed for hypercubes are also applicable in practice to these machines and that multiphase complete exchange can lead to major savings in execution time over traditional solutions.

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, Virginia, 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 overhead is one of the key factors that limit the performance of massively parallel systems. Considerable effort is required to minimize this overhead and no general solutions are as yet in sight. No amount of special hardware or software can eliminate communication overhead. This paper concentrates on the *complete exchange* or *all–to–all personalized* communication pattern. This pattern requires each of a collection of n processors to send a unique message to each of the remaining $n-1$ processors. Complete exchange is required in many important parallel algorithms, such as Fast Fourier Transforms, matrix–vector multiply, the alternating directions implicit (ADI) method for solving partial differential equations, and so on. This is the severest communication requirement that can be imposed on an interprocessor communication network and serves as a useful benchmark of the performance of a parallel computer system.

Prior work on the complete exchange has largely focused on hypercube architectures. Most current commercial multiprocessors are not hypercubes. However, modern machines have powerful interconnection hardware and can be made to *emulate* hypercubes with fair success. We describe the performance of *multiphase complete exchange*, a family of algorithms originally designed for hypercubes, on three contemporary machines: the Intel Paragon, the IBM SP2 and the Meiko CS-2. We discuss the architectures of these machines, present their basic performance parameters and then describe how the multiphase algorithm performs on all three.

2 The Complete Exchange

The complete exchange is a communication pattern that is required in many important applications such as matrix transposition, matrixvector multiply, Fast Fourier Transforms and the Alternating Directions Implicit (ADI) method for solving partial differential equations. To understand the data movement required by this pattern refer to Figure 1 which shows a 4×4 block matrix stored on 4 processors. In part (a) of this Figure the matrix is stored in column order. In part (c) the layout has been changed to row order. It is clear that to change from (a) to (c), each processor must transmit a block of data to every other processor. This is shown in part (b) which is

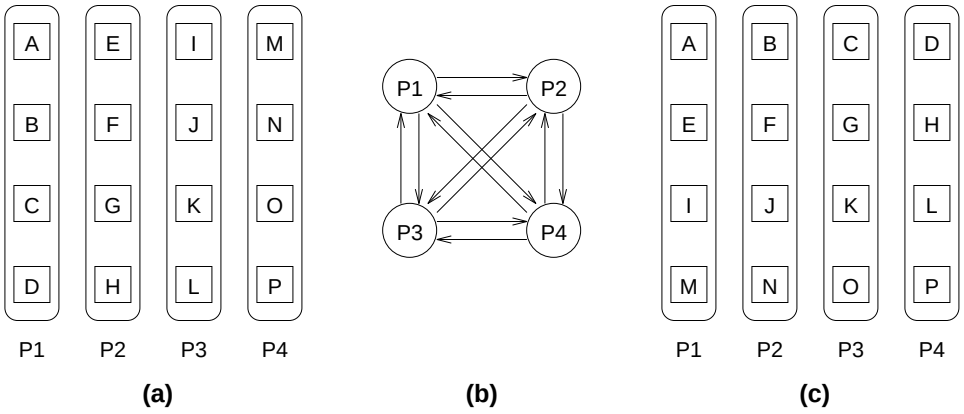

Figure 1: Complete Exchange on 4 Processors. To change storage of blocks from column order (a) to row order (c), each processor must send a distinct message to every other processor (b).

a complete directed graph of four nodes. In general, complete exchange on n processors can be represented by a complete directed graph of n nodes.

Most of the work to date on algorithms for the complete exchange has addressed hypercube architectures. Figure 2 shows a hypercube of dimension $d = 4$ with $n = 2^d = 16$ processors. Each processor is given a binary label and two processors are connected with a communication link if and only if their labels differ in exactly one bit. Each processor in a hypercube is connected to $d - 1$ other processors. As we increase the size of the hypercube, the number of communication links leaving a processor increases logarithmically with the number of nodes. This is the main reason for the difficulty of constructing hypercubes. Nevertheless, hypercubes have enjoyed success since their rich and recursively definable interconnection permits the development of elegant algorithms for communication. The Intel iPSC-2 & 860 and the nCube2 & 3 are examples of commercially produced hypercubes.

Almost all hypercubes use the "e-cube" routing algorithm for moving data between processors. In essence this algorithm moves the message from processor to processor by moving in a direction that successively increases

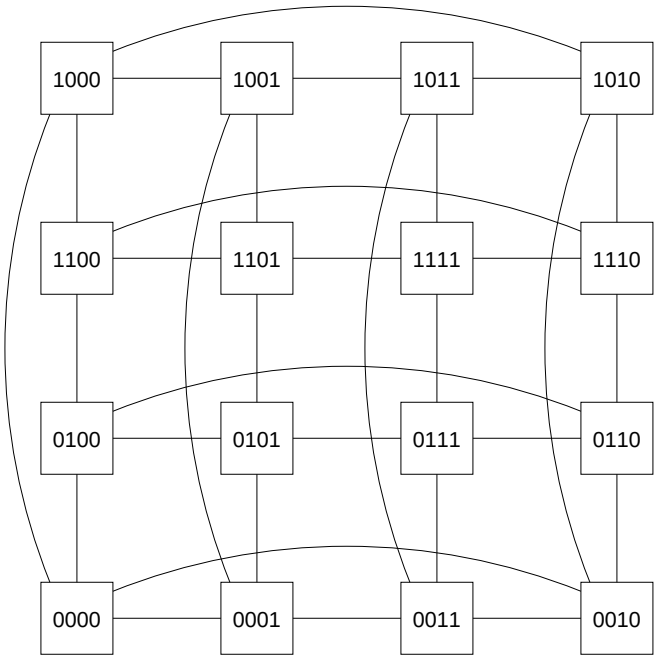

Figure 2: A hypercube of dimension $d = 4$ and size $n = 2^d = 16$. Each node is labeled in binary. Two nodes are connected if their binary labels differ in exactly one bit position.

the match between current processor and the destination. Thus to travel from processor 0010 to 1001, the path taken would be: $0010 \rightarrow 0011 \rightarrow 0001 \rightarrow 1001$. On modern hypercubes, this message transmission is handled by special communications hardware and does not disturb the computations being carried out at intermediate nodes.

The time required to transmit a message from one node to another (assuming no contention for communication links) is modeled by the expression $t = \lambda + \tau m$, where m is the message size in bytes, τ the time per byte (which is the inverse of the communication bandwidth) and λ is the startup overhead, which is due largely to operating system activities required to launch the message. This expression applies equally well to the non-hypercube architectures discussed later in this paper. Over the past decade, improvements in technology have made τ improve from about 0.1μsec to less than 0.01μsec. However, the startup time has remained in the $50 - 100\mu$sec range.

2.1 Standard Exchange

The standard exchange algorithm was developed by Johnsson & Ho [7]. The following pseudo-code executes on each processor while running this algorithm. *mynumber* is the label on each processor, as described in Figure 2. The symbol $\oplus$ stands for bitwise exclusive-or.

```
Standard_exchange{
    for j= d − 1 downto 0 do{
        if (bit j of mynumber = 0)
            message=blocks n/2 to n − 1
        else
            message=blocks 0 to n/2 − 1
        send_message_to_processor((mynumber) ⊕ (2^j))
        shuffle blocks;
    }
}
```

Figure 3 clarifies the operation of this algorithm, which requires a total of $\log n$ transmissions of $n/2$ blocks each. Blocks of data must be permuted between each communication step in order to correctly route them to their destinations. The logarithmic number of transmissions of this algorithm

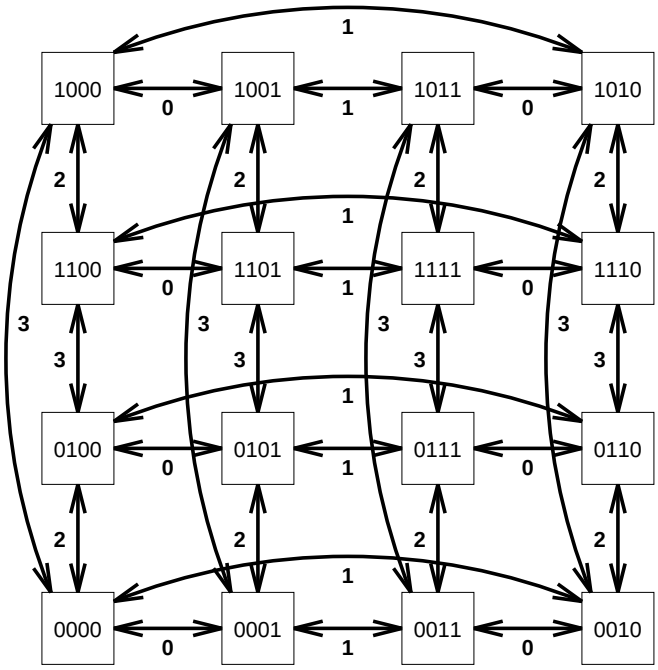

Figure 3: The standard exchange algorithm takes d steps. During the jth step, nodes that differ in bit position j interchange data (indicated by double headed arrows in the figure). The figure shows the entire algorithm with each double headed arrow standing for an interchange of messages between the processors at its endpoints. The label on each arrow is the step in which the exchange is carried out. Since every possible pair of processors does not interchange messages it is clear that messages must be forwarded through intermediate nodes to their ultimate destinations. Shuffling of the blocks is required to route correctly blocks to their destinations.

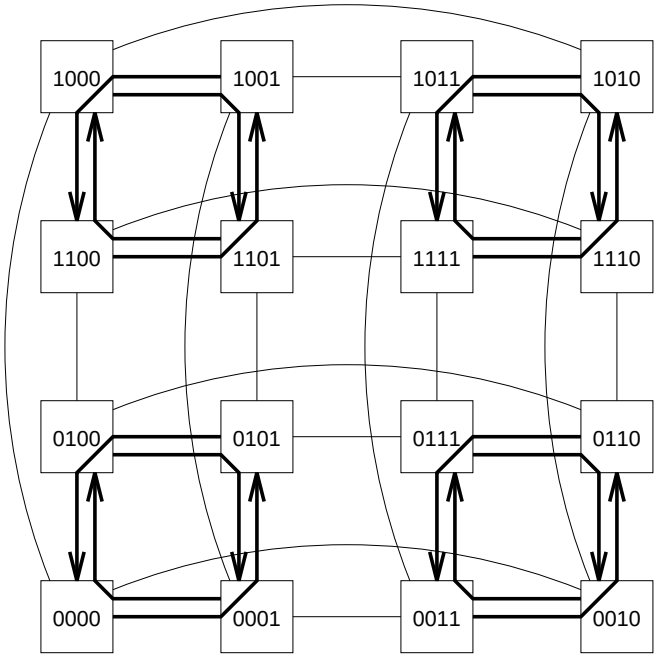

Figure 4: The direct exchange algorithm takes $n-1$ steps. During step i, node j sends a block to processor $i \oplus j$. The figure shows data movement for step $i = 0101$. No data permutation is required in this algorithm as each message block is transmitted directly to its ultimate destination.

reduces the impact of startup time λ (discussed above) and leads to very good performance when message sizes are small.

2.2 Direct Exchange

This algorithm transmits each block *directly* to its ultimate destination (Figure 4). It was originally published by Take [10]. Subsequent work on implementing it on the Intel iPSC-860 hypercubes was carried out by Seidel [9] and Bokhari [4].

```
Direct_exchange{
    for i= 1 to n − 1 do
        send_block_to_processor((mynumber) ⊕ (i))
}
```

This algorithm is asymptotically optimal in that it requires exactly $n-1$ messages of one block each to achieve the complete exchange. It is always the best algorithm to use for very large message sizes. The deceptively simple exclusive-or schedule guarantees that there is no contention for communication links under the "e-cube" routing strategy. The fact that each block is transmitted directly to its destination means that there is no shuffling overhead.

2.3 Multiphase Complete Exchange

Multiphase complete exchange is a family of algorithms that compromises between the starting overhead of direct exchange and the shuffling and data transmission overhead of standard exchange. It was developed by Ho & Raghunath [6] and subsequently investigated by Bokhari [2]. Figure 5 describes the operation of this algorithm.

A detailed exposition and analysis appears in [3, 8], where it is shown that each partition of the integer d (the dimension of the hypercube) leads to a multiphase algorithm for complete exchange. For example, for $d = 5$ the partitions are $\{1,1,1,1,1\}$, $\{1,1,1,2\}$, $\{1,2,2\}$, $\{1,1,3\}$, $\{1,4\}$, $\{2,3\}$ and $\{5\}$. In this set of partitions, $\{1,1,1,1,1\}$ corresponds to standard exchange and $\{5\}$ to direct exchange. Theory developed in [3, 8] shows that of the set of partitions of d only *equipartitions* (partitions in which the largest and smallest element differ by at most 1) can ever be optimal. Thus, for $d = 5$ the optimal multiphase algorithms are those corresponding to $\{1,1,1,1,1\}$,$\{1,2,2\}$,$\{2,3\}$ and $\{5\}$. It can be proved that the number of these optimal partitions is no more than $2\sqrt{d}$. This is a very small number since d, the dimension of the hypercube, equals $\log n$, where n is the number of nodes. Figure 6 shows the run times of the family of multiphase algorithms plotted against message size for a hypothetical hypercube of dimension $d = 5$. Some algorithms have run times that are never optimal and are of no interest to us. The three algorithms of interest are the ones corresponding to the partitions $\{1,1,1,1,1\}$, $\{2,3\}$ and $\{5\}$ because these are the ones that are optimal.

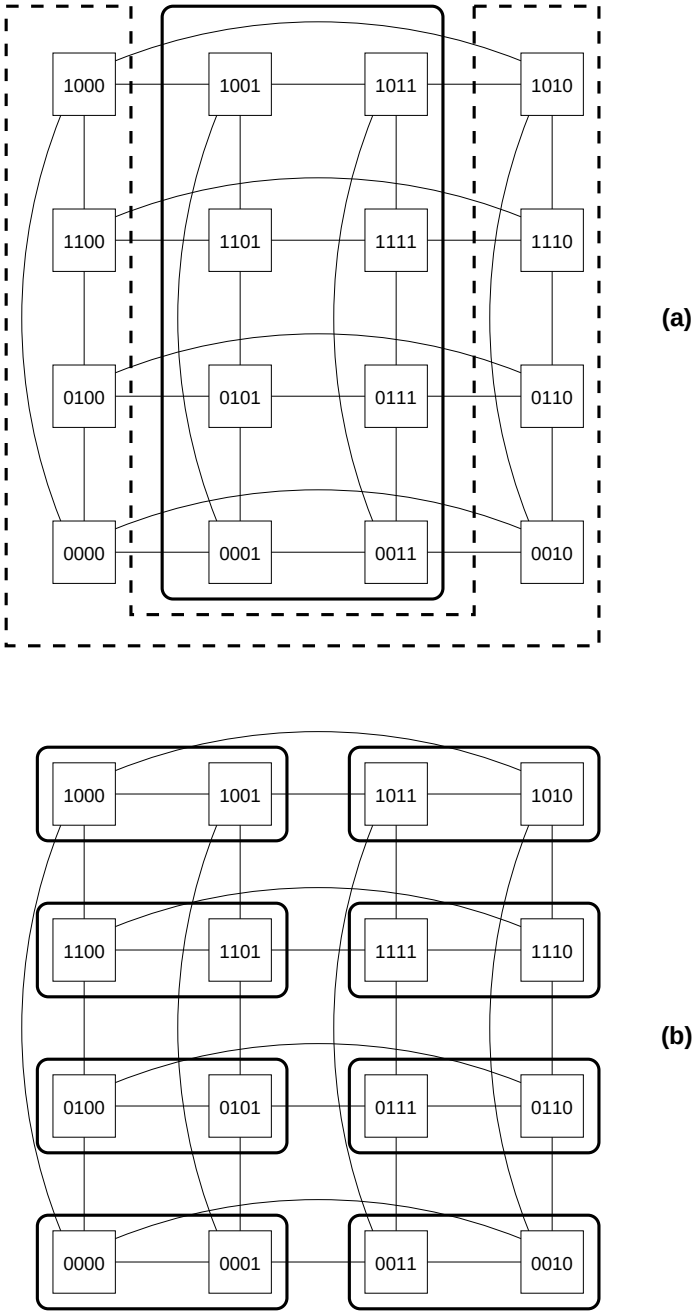

Figure 5: A multiphase algorithm on a 16 node hypercube. (a) Shows direct exchanges being carried out separately on two 8 node subcubes. This is followed (b) by direct exchanges on 8 two node hypercubes. A data permutation step is required between (a) and (b) to correctly route the data.

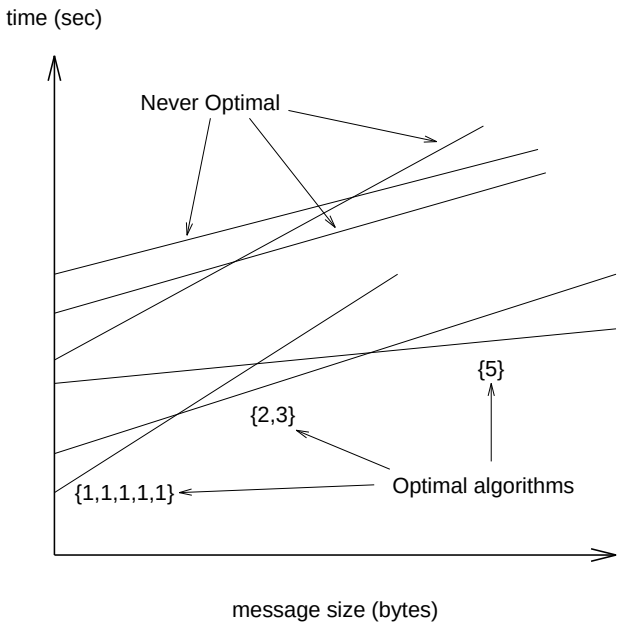

Figure 6: Only multiphase algorithms corresponding to equipartitions can ever be optimal. The figure shows what can happen when $d = 5$.

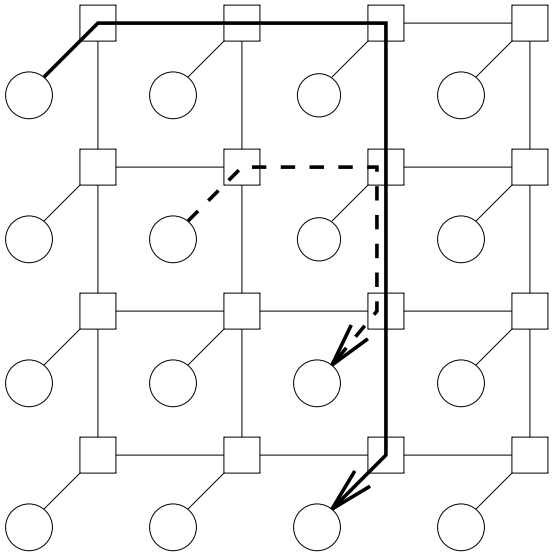

Figure 7: The mesh interconnect of a 4×4 Paragon. The circles represent compute nodes while the squares show special purpose hardware for communication. Message routing is done via the "row–column" algorithm explained in the text. The figure shows two pairs of processors communicating and contending for a single edge. Such *edge contention* can lead to substantial overhead.

3 The Architectures

The machines on which we evaluated the multiphase complete exchange are the Intel Paragon, IBM SP2 and Meiko CS-2. All three machines are in commercial production and incorporate special purpose hardware for interprocessor communication.

3.1 Intel Paragon

The Intel Paragon on which the experiments described in this report were carried out is located at the Center for Advanced Computing Research at Caltech. It is a mesh-connected machine with 512 processors arranged in a 32×16 rectangle. Each processor is connected to four neighbors through special purpose hardware (Figure 7). Each node on this machine has two

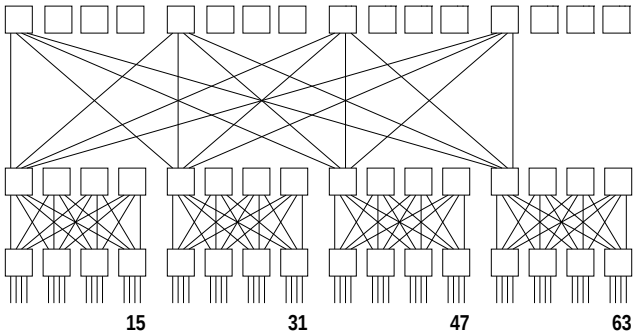

Figure 8: A multistage interconnect of the type used in the SP2 or CS-2. Each square represents a 4×4 bidirectional crossbar switch. Any two processors can be connected to each other by suitably setting the switches. Most of the connections leading into the topmost layer have been omitted to avoid a congested diagram.

Intel i860 processors (one for computation and one for communication) and 32 MBytes of memory. The i860s run at 50MHz and are capable of 75 MFlops each. Programming on this machine was done in C augmented with the `nx` message passing library. This library permits programs to send and receive messages from other processors, carry out global synchronization, compute global sums, etc., via calls to C functions. Message routing on this machine is done using the "row–column" rule. A message first travels along a row until it reaches the column on which the destination lies. It then travels along the column until it reaches the destination.

3.2 IBM SP2

The Cornell Theory Center's 512 node IBM SP2 multicomputer was used for these experiments. The processors on this machine are interconnected through a multistage switch (Figure 8). Each square box in this figure represents a bidirectional 4×4 switch. In theory each processor can talk to any other without contention for switches or links. In practice the setting up of such connections is difficult to implement on the fly and significant degradation due to contention is seen. Each computational node (not shown in Figure 8) has a POWER2 architecture RS/6000 processor that runs at 66.7

MHz, has at least 128 MBytes of memory, and is capable of 266 MFlops. This machine was programmed in C using the `MPI` message passing library [5]. `MPI` provides roughly the same functions on the SP2 as the `nx` library does on the Paragon.

3.3 Meiko CS-2

The Vienna Center for Parallel Computing has recently installed a Meiko CS-2. This is a 128 processor machine interconnected through a multistage switch similar to that of the SP2 (Figure 8). Each node is a SuperSPARC running at 50MHz, with 64 MBytes of memory and capable of 100 MFlops. This machine was programmed in C using the `mpsc` library which is designed to be fully compatible with the `nx` library on the Intel hypercubes and the Paragon.

While all three machines incorporate powerful interprocessor communication mechanisms, the programmer still has to take many factors into account in order to implement efficient parallel algorithms. These issues are discussed in detail by Bokhari [1].

4 Performance Measurements

There are 3 key performance figures of a parallel machine that determine its success at executing multiphase complete exchange. These are

Communication time: the time required to send a message of m bytes from one processor to another.

Synchronization time: the time for the machine to execute a barrier (that is, to ensure that all processors have reached a specified point in the parallel program.) This is important because multiphase complete exchange requires data transfers to be carefully scheduled for correct operation.

Memory copy time: Excluding the purely *direct* algorithm, all multiphase algorithms require some amount of data permutation *within* a single processor in order to route data blocks to their correct destination. Thus, memory-to-memory transfer time within the same processor is an important measure of performance.

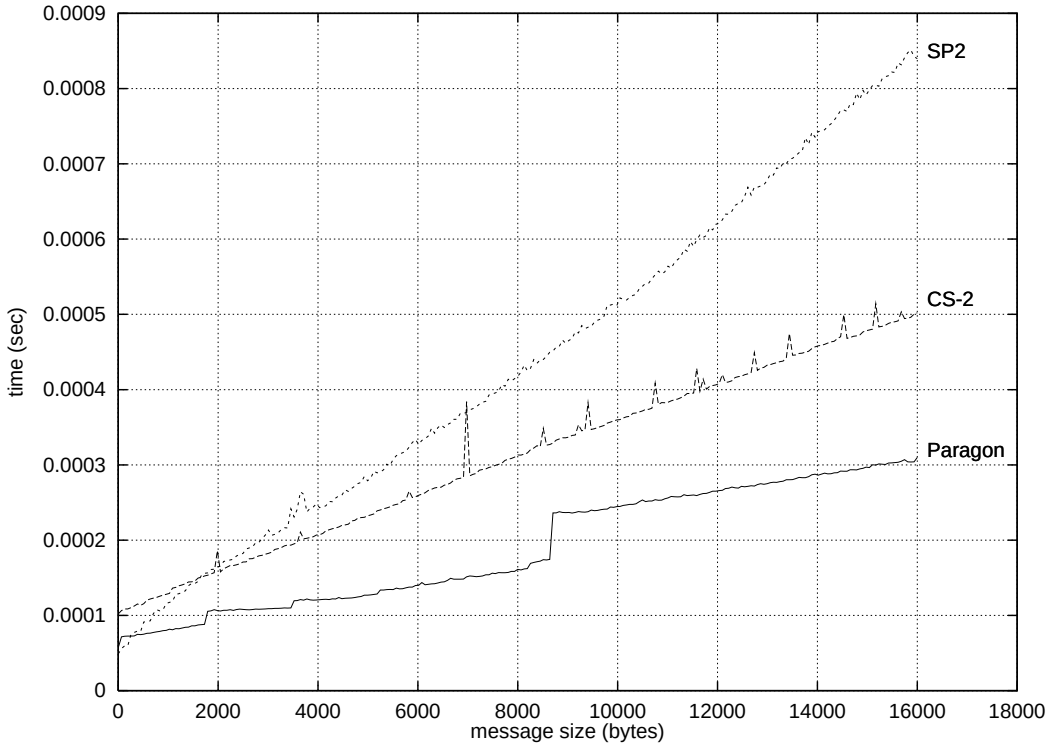

Figure 9: Communication time on the Paragon, SP2 and CS-2.

Figure 9 shows the communication time for all three machines, measured over the range 0 to 16000 bytes in increments of 64 bytes. The discontinuities in the Paragon plots are caused by packetization overhead. The spikes on the plots for the SP2 and CS-2 are caused by interference from other jobs or by operating system events. Table 1 summarizes this information and also includes measurements of synchronization and memory copy time. The expressions for run times are for messages smaller than 8000 bytes, as this is the range of interest to us as far as the multiphase complete exchange is concerned.

5 Experimental Measurements

Figures 10 and 11 show the performance of the multiphase complete exchange on 32 and 64 processor pools on the three machines under study. On

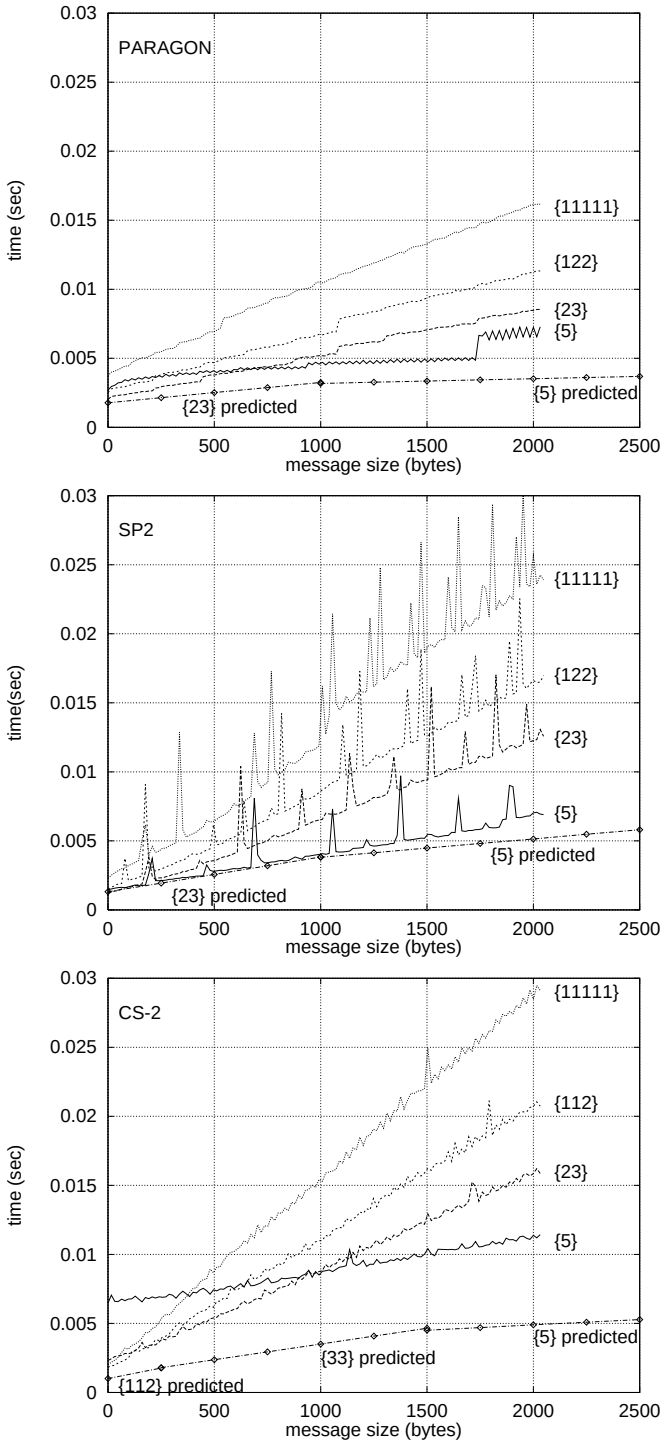

Figure 10: Performance of multiphase complete exchange on 32 processors.

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

Table 1: Summary of performance figures

the Paragon we use 4×8 and 8×8 submeshes while on the SP2 the allocation of processors is beyond our control. On the CS-2 we obtained measurements over contiguously numbered sets of processors.

The plots obtained have the general shape predicted by the theory of [3, 8]. The direct algorithms $\{5\}$ and $\{6\}$ are optimal for large message sizes. The standard algorithms $\{1, 1, 1, 1, 1\}$ and $\{1, 1, 1, 1, 1, 1\}$ tend to have good performance for very small message sizes. The algorithms corresponding to equipartitions of cardinality 2, that is $\{2, 3\}$ and $\{3, 3\}$ are always optimal for small message sizes. This is very similar to the results for the Intel iPSC-860 hypercube given in [2].

In Figures 10 and 11 we have also plotted the *predicted* run time of the two best algorithms based on the performance figures given in Table 1 and the formulae in [3]. The agreement here is very poor and the predicted plots serve only to give a qualitative idea of the shape of the measured plots. This is because the predicted curves assume a hypercube interconnect which can execute the multiphase algorithm without any contention for communication links. Our machines are not hypercubes and suffer from link and switch contention. Nevertheless these plots show the benefits of adopting the multiphase approach.

The noise or fluctuation in the plots for the SP2 are particularly noteworthy. We believe this to be caused by contention for switches by jobs other than our own job. Very wide fluctuations are encountered on the SP2, making the task of predicting performance very difficult.

The intensity of the complete exchange communication pattern stresses communication hardware and software very severely. On the SP2 we were unable to run successfully beyond 64 processors because of switch problems

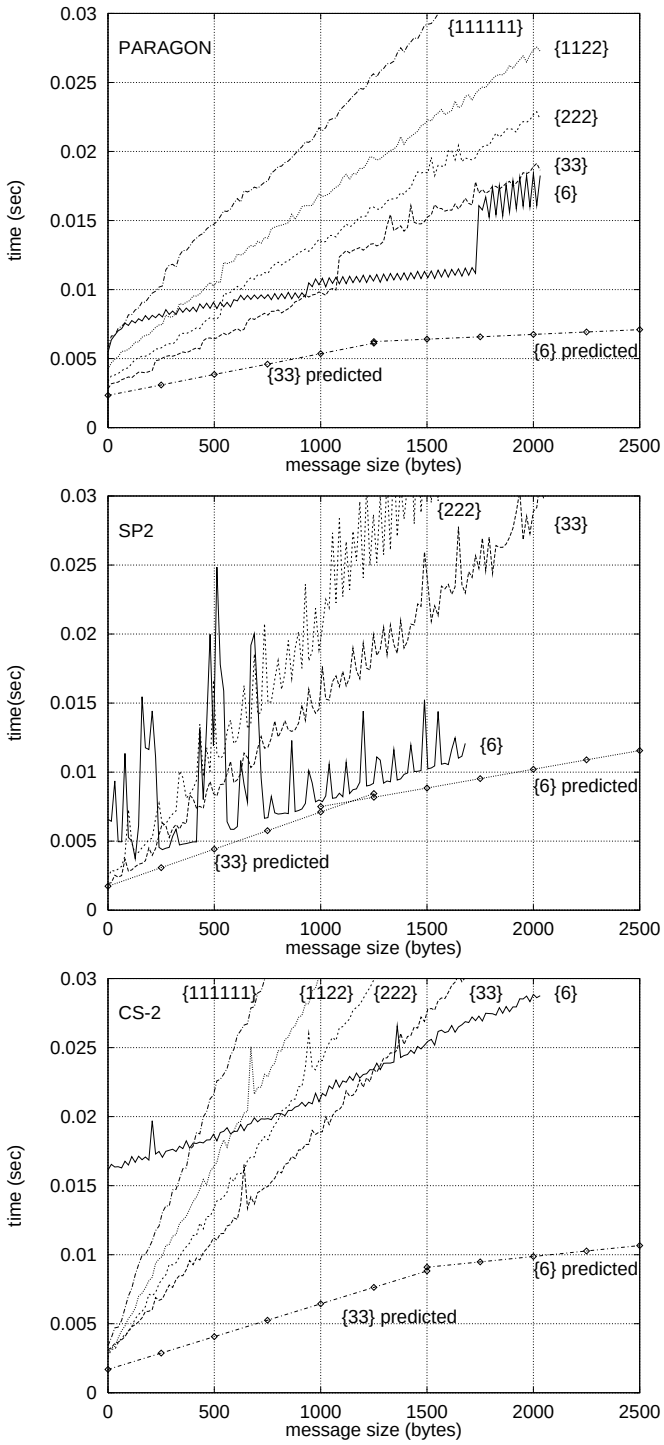

Figure 11: Performance of multiphase complete exchange on 64 processors.

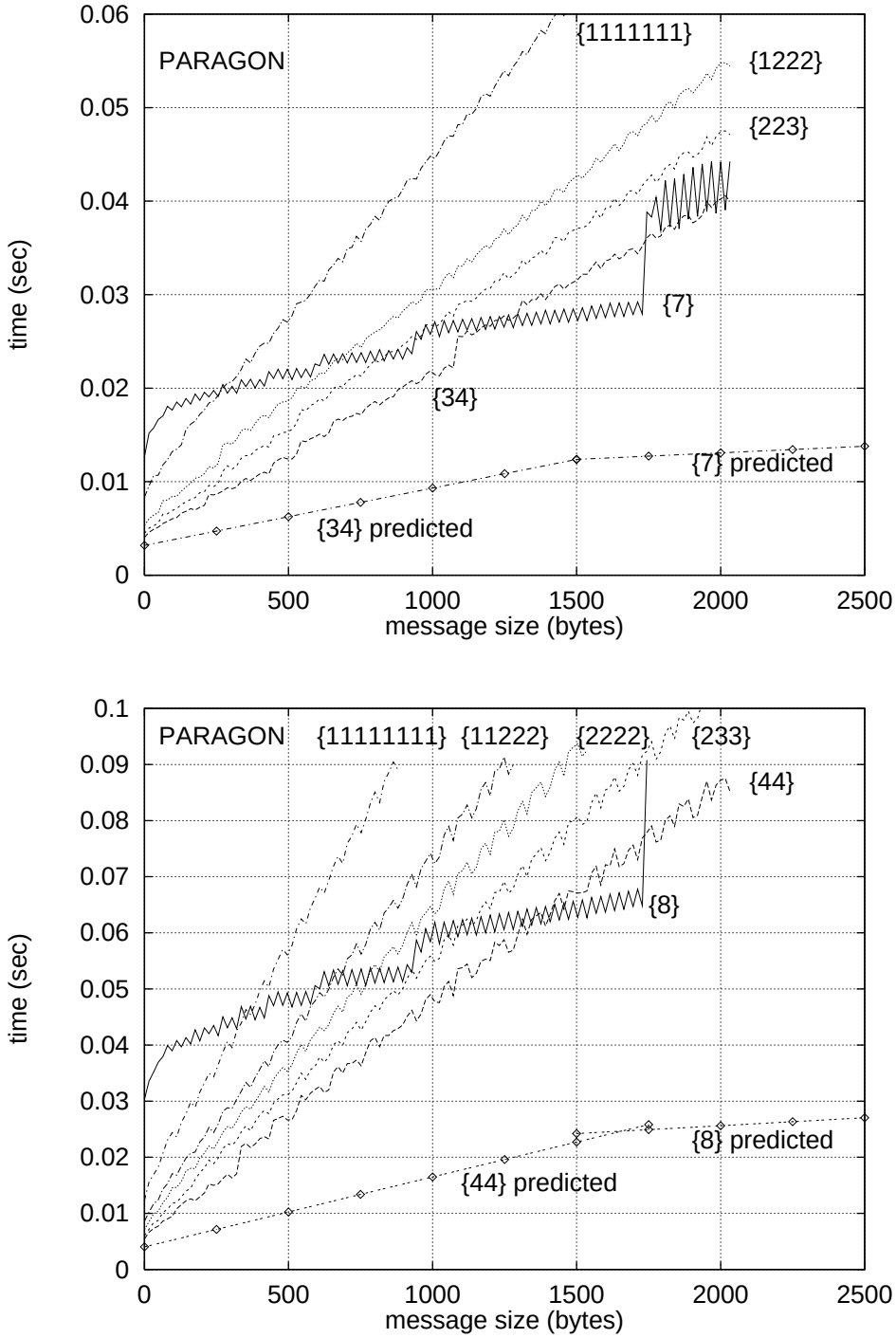

Figure 12: Multiphase complete exchange on 128 and 256 processors on Paragon.

presumably caused by intense communication. On the Paragon, although we were able to run on submeshes as large as 16×16, the operating system could not accommodate the 255 message receives required by the direct algorithm {8} for the entire range of message sizes. The plot for this algorithm in Figure 12 stops abruptly at 1728 bytes for this reason.

These experiences, though unpleasant, underline the utility of multiphase complete exchange as a "stress test" of communication hardware and software. We are confident that the problems encountered will be resolved by the respective manufacturers in due course.

6 Conclusions

Interprocessor communication is what makes parallel programming challenging. This paper has explored the performance of three contemporary parallel machines when carrying out the complete exchange–the densest communication pattern possible. We have shown that the multiphase complete exchange family of algorithms, which were originally developed for hypercubes, perform well on modern non-hypercube machines.

The performance of multiphase exchange on these machines does not match well the figures predicted from basic performance parameters. This is because there are complex effects of link contention, switch contention, paging disturbance and overheads due to operating system timer interrupts on these machines that are not captured by the basic parameters. Furthermore, although these machines can execute hypercube algorithms with good performance, they are really not hypercubes and thus suffer from a mismatch of the algorithm to the architecture. This observation demonstrates the falsity of the commonly held belief that, in modern parallel machines, the matching of algorithm to architecture is irrelevant. If that had been the case, these machines would have given us predictable performance, as is the case with hypercube implementations of the same algorithms [2].

The complete exchange problem is severe enough to have uncovered several problems with the communication hardware and software of two of the machines studied. This points out the utility of using it as an extremely stressful test of parallel architectures.

Future work in this area should address the problem of designing exchange algorithms that take the specific architectures of these and other modern

machines into account. It would also be useful to study the Paragon, SP2 and CS-2 in greater detail, so that a more precise performance model can be developed. Such a model will be invaluable in permitting practitioners to evaluate the efficiency of their parallel algorithm implementations.

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, David Nicol, John van Rosendale and Linda Wilson. 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 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, 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 to me during my work on the SP2.

References

[1] S. H. Bokhari. Communication overhead on the Intel Paragon, IBM SP2 and Meiko CS-2. ICASE Interim Report 28, September 1995.

[2] S. H. Bokhari. Multiphase complete exchange on a circuit switched hypercube. In *Proc. 1991 International Conf. Parallel Processing*, pages 525–529, 1991.

[3] S. H. Bokhari. Multiphase complete exchange: A theoretical analysis. Technical Report 93-64, ICASE, August 1993. NASA Contractor Report 191531. To appear in *IEEE Trans. Computers*.

[4] S. H. Bokhari. Complete exchange on the Intel iPSC-860 hypercube. Technical Report 91-4, ICASE, January 1991.

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

[6] C-T. Ho and M. T. Raghunath. Efficient communication primitives on hypercubes. In *Proc. 6th. DMCC*, pages 390–397, 1991.

[7] S. L. Johnsson and C-T. Ho. Matrix transposition on boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988.

[8] David Nicol and Shahid Bokhari. Optimal multiphase complete exchange on circuit-switched hypercube architectures. In *Proc. 1994 ACM SIGMETRICS Conf*, pages 252–260, Nashville, TN.

[9] Steve Seidel, Ming-Horng Lee, and Shivi Fotedar. Concurrent bidirectional communication on the Intel iPSC/860 and iPSC/2. Technical Report CS-TR 9006, Dept. of Computer Science, Michigan Tech. Univ., November 1990.

[10] R. Take. A routing method for the all-to-all burst on hypercube network. In *Proc. 35th. National Conf. Info. Proc. Soc. Japan*, pages 151–152, 1987. In Japanese.

A Related Web Sites

Additional information about the computing centers and computer architectures described in this report can be found at the following web sites.

Caltech Center for Adv. Comput. Res.	`http://www.ccsf.caltech.edu`
Intel Paragon	`http://www.ssd.intel.com/paragon.html`
Cornell Theory Center	`http://www.tc.cornell.edu`
IBM SP-2	`http://ibm.tc.cornell.edu/ibm/pps/sp2/`
Vienna Center for Parallel Computing	`http://www.vcpc.univie.ac.at/vcpc.html`
Meiko CS-2	`http://www.meiko.com/`
ICASE	`http://www.icase.edu/docs/home.html`