

Portable and Scalable Algorithms for Irregular All-to-All Communication *

Wenheng Liu, Cho-Li Wang and Viktor K. Prasanna
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-2562

Abstract

In this paper, we develop portable and scalable algorithms for performing irregular all-to-all communication in High Performance Computing (HPC) systems. To minimize the communication latency, the algorithm reduces the total number of messages transmitted, reduces the variance of the lengths of these messages, and overlaps the communication with computation. The performance of the algorithm is characterized using a simple model of HPC systems. Our implementations are performed using the Message Passing Interface (MPI) standard and they can be ported to various HPC platforms. The performance of our algorithms is evaluated on CM5, T3D and SP2. The results show the effectiveness of the techniques as well as the interplay between the architectural features, the machine size, and the variance of message lengths. The experiences of our study can be applied in other HPC systems to optimize the performance of collective communication operations.

1. Introduction

High performance computing (HPC) systems such as IBM SP2, Cray T3D, among others, and Networks of workstations (NOWs) [1] are being employed to solve large scale scientific and engineering problems. In these platforms, the computing nodes are very powerful; they can be individual workstations or even Massively Parallel Processors (MPPs). However, the communication latencies can be very high, particularly in the case of loosely-coupled platforms using message passing. Many design issues at the architectural level affect the communication latency. These include network topology, the available network bandwidth, packet routing schemes, network interfaces, and protocols employed.

* This research was supported in part by ARPA under contract DABT63-95-C-0092. Cho-Li Wang was supported in part by NSF under grant CCR-9317301. His current address is Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong. The authors can be reached by email at {liu+prasanna}@halcyon.usc.edu and clwang@cs.hku.hk.

While parallelizing many scientific and engineering applications in various fields, the communication among the nodes is often irregular [2]; i.e., the length of the messages exchanged vary from node to node. The major factors that affect the performance of irregular communication are large start-up latency in message passing and possible node contention. Node contention can be a serious problem due to the variance in the message lengths.

In this paper, we develop an architecture-independent algorithm for irregular all-to-all communication. There are four stages in our algorithm. The first two stages perform message distribution, while the rest of the stages perform message collection. In each stage, an all-to-all communication is performed along with some local memory accesses to compose these messages. The algorithm reduces node contention by balancing the length of the messages transmitted in each stage. It also reduces the latency by reducing the number of message initiations. Variations of the algorithm can be obtained by overlapping the computation with the communication. Our initial work in designing these algorithms appear in [13]. That work was motivated by the need to perform irregular and data dependent communication operations in parallelizing intermediate and high level vision problems.

Recently, algorithms have been proposed by Ranka [4] and by Hambrush [3] for communication operations. The algorithm in [4] reduces node contention and has been implemented on CM5 using active messages. In this algorithm, the number of message passing start-ups is doubled; it is efficient when the traffic is large. The algorithms in [3] are motivated by mesh architecture and the messages are transmitted in nonblocking mode. The node contention problem can degrade these performance of the algorithm and affect their scalability.

We compare the performance of our algorithm against a straight-forward single-stage algorithm and the two prior algorithms mentioned above. We employ a simple and realistic communication model to analyze the performance of these algorithms. The scalability of our algorithm is shown based on the analysis.

The experimental evaluations were conducted on CM5, SP2, and T3D. The results show the effectiveness of the techniques: reducing the number of messages communicated is the most efficient way to improve performance. Reducing the irregularity of the messages causes significant data access overhead; benefits will be noticeable only when the local buffers can be accessed relatively quickly and there is a large variance in the message sizes that cause node contention. Overlapping of communication with computation is effective on SP2, and T3D due to their architectural support for performing communication. We have implemented the algorithms using the Message Passing Interface (MPI) primitives. Our techniques can be further exploited at the lower level (operating system and network interface level) to optimize the performance of collective communication operations.

The rest of the paper is organized as follows. Section 2 describes the overheads in performing irregular communication. Section 3 describes the algorithms and their analysis. Section 4 describes the experiments and the results on CM5, T3D, and SP2. Section 5 discusses our results and concludes the paper.

2. Overheads in Performing Irregular Communications

In this section, we first define irregular communication problems and then identify the overheads in performing these operations in HPC systems.

2.1. Irregular Communication

In irregular many-to-many communication, some nodes concurrently read from and write to some other nodes and the size of the messages transmitted varies from one to the other. Such communications arise in scientific computations in environmental sciences, [16], in high level vision problems [14, 13], and in network simulations [11]. In [11], irregular problems arising in scientific and industrial applications have been classified into loosely synchronous, asynchronous, embarrassingly parallel, and meta-problem categories; these applications result in several irregular communication operations.

The main problem encountered during irregular communication is node contention. Node contention occurs when messages compete for buffers at the network interfaces, and the interfaces cannot handle all of the messages at the same time [12]. Figure 1 illustrates the node contention problem in irregular communication assuming that each processor has a single port.

At time T1, nodes P0, P1, P2, and P3 start sending a message to P1, P2, P3, and P0 respectively. Assume that P0 completes its transmission at time T2 and starts sending its next message to P2. At this time, there is a potential node

contention at P2. Similarly, at time T3, another potential node contention occurs at P0 since both P2 and P3 attempt to send messages to P0 concurrently.

2.2. Communication Overheads

Figure 2 illustrates the steps in message passing in a typical high performance system. The message passing latency is the time spent to initiate a communication, generate, send, and receive a message. Messages are copied from the local memory space to the network interface buffer at the sending node and from the interface buffer to the local memory at the receiving node. The memory copies can be performed either by the main processor or by direct memory access (DMA). The latencies illustrated in Figure 2 are defined as follows.

- T_{d_send} and T_{d_recv} : start-up latencies for message passing initiation at the sending and receiving nodes respectively.
- t_{m_send} : message coalescing latency per byte.
- t_{m_recv} : message decomposing latency per byte.
- t_{d_send} and t_{d_recv} : memory copy latencies per byte between local memory and interface buffer at sending and receiving nodes respectively.
- t_d : message transfer time per byte; time spent by the message in the communication channel.

Based on Figure 2, a simple estimate of the total time consumed to send and receive M messages with lengths L_i bytes, $i=0..M-1$, is:

$$T = M \times T_d + \left(\sum_{i=0}^{M-1} L_i \right) \times t_m + \left(\sum_{i=0}^{M-1} L_i \right) \times f_{nc} \times t_d \quad (1)$$

T_d is the start-up latency per message; this includes T_{d_send} and T_{d_recv} . t_m is the message access latency per byte including t_{m_send} , t_{m_recv} , t_{d_send} , and t_{d_recv} , while t_d is the transmission latency per byte. f_{nc} is an aggregate node contention factor, which equals 1 if there is no node contention.

Equation 1, even though simplistic, illustrates two problems that can arise in performing irregular communication: number of message start-ups (the first term in Equation 1), and node contention. The communication features the state-of-the-art HPC platforms such as CM5, SP2, T3D, and workstation clusters interconnected by an ATM network or Myrinet are shown in Table 1. The table is based on [5, 6, 7, 8, 9, 10] and our own measurements. It should be noted that the numbers vary depending on the version of the software environment used for message passing.

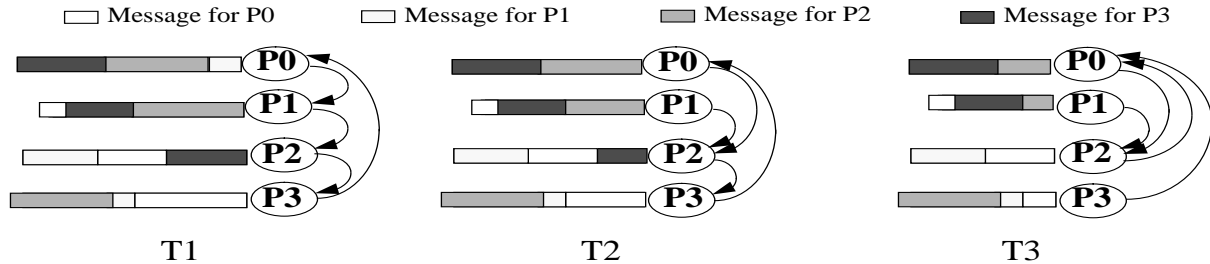


Figure 1. A scenario depicting possible node contentions in performing irregular communication

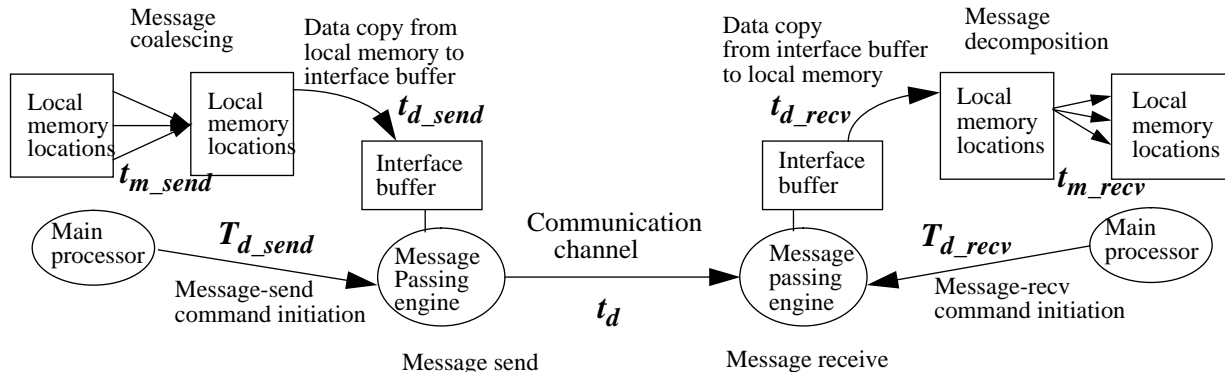


Figure 2. Steps in message passing in a typical high performance parallel or distributed system

Table 1: Communication features of HPC platforms

Platform	I/O Ports	Message Passing Processor	Message Passing DMA	$T_d(\mu\text{sec})$	$t_d(\frac{\mu\text{sec}}{\text{byte}})$
CM5	Single I/O port	Main processor	No	170	0.12
ISP2	Bidirectional I/O ports	Co-processor	Yes	96	0.035
T3D	Bidirectional I/O ports for a pair of nodes	Supplementary circuit	Yes	186	0.043
Workstations + ATM network + SBA-200	Individual I/O ports and send/recv buffers	Intel i960 dual-issue Control processor	Yes	~240	0.14
Workstations + Myrinet + LANai interface	Single I/O port	Microprocessor in LANai board	Yes	~600	~0.08

3. Algorithms and Analysis

An all-to-all personalized communication, also known as total exchange, involves exchange of messages between every pair of processors; each node sends a distinct message to every other node. We were originally motivated by parallelizing problems arising in intermediate and high level vision. A preliminary version of our algorithm for irregular communication appears in [13]. To evaluate the performance of our algorithm, we have studied four different algorithms, denoted A1 through A4. A1 is a straightforward algorithm. A2 and A3 are Ranka's [4] and Hambrusch's [3] algorithms. A4 is our four stage algorithm. For the sake of completeness, we describe A1, A2, and A3 before describ-

ing A4. Throughout this paper, P denotes the number of processing nodes.

3.1. Algorithm A1

A straightforward algorithm for irregular all-to-all personalized communication among P nodes is a one-stage algorithm. It operates in $(P-1)$ iterations. During the i th iteration, $1 \leq i \leq P-1$, processor j sends $((i+j) \bmod P)$ th message to node $(i+j) \bmod P$. The operation of the nodes is synchronized at the beginning of each communication iteration. The node contention problem described in Section 2 can severely degrade the performance of this algorithm when there is a large variance in the message sizes.

3.2. Algorithm A2

Ranka, Shankar, and Alsabti proposed a two-stage algorithm [4] that decomposes a collective communication with high message size variance into two collective communication stages with low message size variance. The main consideration of the algorithm is to reduce node contention by smoothing out the variance of the message size.

In the first stage of the algorithm, at each processing node, each of the P messages are evenly divided into P slices. The slices for all destinations are coalesced to generate P equal-sized intermediate messages. Each node distributes $P-1$ of these messages to the other $P-1$ nodes. In the second stage, the received data are rearranged at each node

and sent to their destinations; each node sends $P-1$ messages in the second stage. Their implementations were performed on CM5 using active messages.

3.3. Algorithm A3

Hambrusch, Hameed and Khokhar have considered several collective communication operations. They have proposed another two-stage algorithm [3] for all-to-all communication which combines the actual messages into longer messages and redistributes them before sending them to their destination.

They assume a square mesh of size $\sqrt{P} \times \sqrt{P}$ with a two dimensional indexing scheme. The P nodes are partitioned into \sqrt{P} groups based on the first and second dimensions. In each stage, each node sends $\sqrt{P} - 1$ messages to the other nodes in the same partition. The algorithm reduces the total number of messages by partitioning the P nodes into \sqrt{P} groups (assuming \sqrt{P} is an integer). The goal of the two-level algorithm is to achieve better performance for short messages. Since the actual messages are coalesced into longer messages, the variance of the message sizes is smoothed out in some cases and node contention can be reduced.

P : Total number of nodes, $g: \lceil \sqrt{P} \rceil$
 $G(Par, i)$: The group node i belongs to, based on Partition Par
 $\|G(Par, i)\|$: Number of nodes in group $G(Par, i)$

For two nodes i_1 and $i_2, 0 \leq i_1, i_2 < P$,

Partition A: $G(A, i_1) = G(A, i_2)$, if $\lfloor i_1/g \rfloor = \lfloor i_2/g \rfloor$,
 $Rank(A, i) = i \bmod g$

Partition B: $G(A, i_1) = G(A, i_2)$, if $i_1 \bmod g = i_2 \bmod g$,
 $Rank(B, i) = \lfloor i/g \rfloor$

Partition C: All nodes from 0 to $P-1$

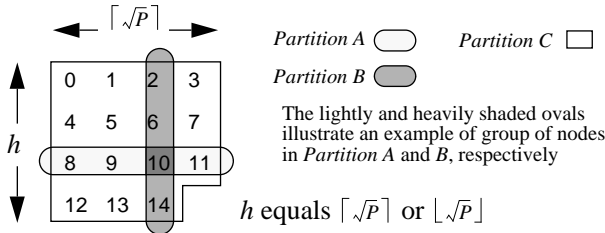


Figure 3. Definitions and notations of node partitions

3.4. Algorithm A4

We develop a four-stage algorithm. It reduces the variance of the message sizes by decomposing the messages into equal slices and reduces the number of message passing start-ups by sending them to small number of interme-

diated nodes. We first define three different partitions of the nodes. These are shown in Figure 3. The primitives for the algorithm are shown in Figure 4.

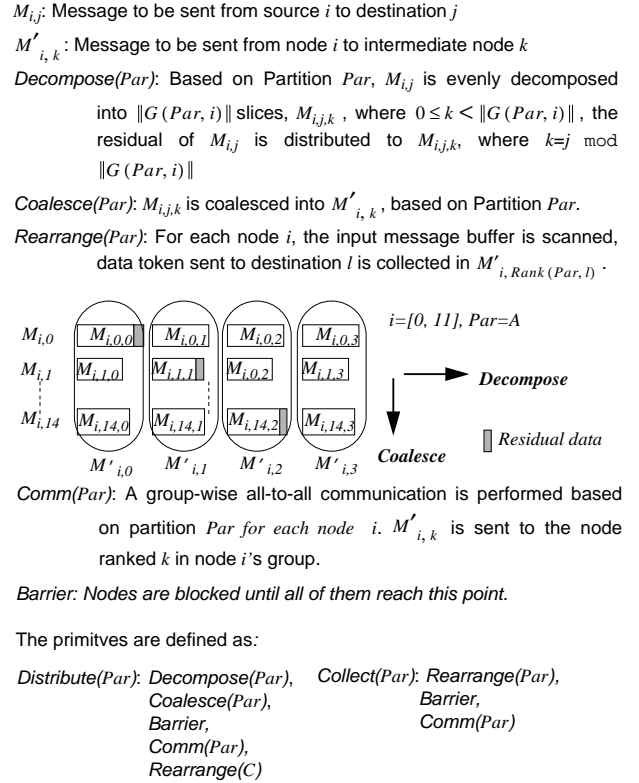


Figure 4. Primitives used in Algorithm A4

Our four-stage algorithm is performed as:

- **Stage 1:** $Distribute(A)$
- **Stage 2:** $Distribute(B)$
- **Stage 3:** $Collect(A)$
- **Stage 4:** $Collect(B)$

After the first two stages of communication, the message sizes have been balanced and the messages are evenly distributed to all of the nodes. Then, the $Collect$ primitive is performed with respect to $Partition A$ and B in the last two stages to cluster the data to their destination nodes. If \sqrt{P} is not an integer, the algorithm can be modified by redistributing some of the intermediate messages without increasing the communication complexity. Details can be found in [15]

Each processor performs only $\lceil \sqrt{P} \rceil - 1$ communication steps in each of the stages. Each communication step is performed to transmit a coalesced message to a destination. The length of the message transmitted between any pair of processors is at most $\lfloor \frac{L}{\sqrt{P}} \rfloor + P$, where L denotes the maximum total traffic among the nodes. The second term reflects

the aggregation of at most $\lceil \sqrt{P} \rceil$ residual messages since the residual data are alternately distributed to the coalesced messages

As depicted in Equation 1 in the previous section, the total latency for message passing can be classified into start-up latency, memory access latency, and transmission latency. Table 2 illustrates the communication complexity of the four algorithms. Note that the transmission latency depends on the length of the messages transmitted and also on the node contention. Table 3 shows the features of the algorithms. It also illustrates the effectiveness of these algorithms.

From Table 2 and Equation 1, our algorithm A4 takes time $(4L + P\sqrt{P})t_d + 4(\sqrt{P} - 1)T_d + 5Lt_m$. The data transmission dominates the start-up latency and local memory access latency when L is $\Omega(\sqrt{P})$. Note that $\Omega(\sqrt{P})$ is a lower bound in the communication.

Table 2: Communication complexity

Algorithm	Memory Access	Start-up Latency	Total message communication
A1	constant	$P-1$	L
A2	$2L$	$2(P-1)$	$2L$
A3	$2L$	$2(\sqrt{P} - 1)$	$2L$
A4	$5L$	$4(\sqrt{P} - 1)$	$4L$

Table 3: Characteristics of the four Algorithms

Algorithm	Reduction of number of messages	Reduction of message irregularity	Comments
A1	No	No	well suited if number of nodes is small
A2	No	Yes	suitable if traffic is equal at each node, and message size is large
A3	Yes	Partially	suitable if message size is small
A4	Yes	Yes	---

We can also partially overlap the communication with computation in implementing our algorithm. For instance, *Rearrange* and *Decompose* primitives can work concurrently with *Comm* primitive. The See Section 4.3 for details.

4. Experiments and Results

The algorithms were implemented in C using the MPI primitives, $MPI_Send()$, and $MPI_Recv()$ for blocking, $MPI_Isend()$, $MPI_Irecv()$, and $MPI_Wait()$ for non-blocking transmissions. We implemented algorithms A1 through A4 on CM5, SP2 and T3D. In the following, l_{max} denotes the longest message (in terms of number of tokens) sent from a processor. A token is an abstract data type, which is 22-byte long in all the machines considered here.

Three different communication patterns were used for the experiments. They are described as follows:

- Pattern1: Each node i sends l_{max} tokens to destination $(i+1) \bmod P$, and one token to the rest of the nodes.
- Pattern2: Each node i sends l_{max} tokens to destination $(i+1) \bmod P$, and destination j , if $j \bmod \lceil \sqrt{P} \rceil = 0$, and one token to the rest of the nodes.
- Pattern3: Each node sends a random number of tokens (between 1 and l_{max}) to each of the other nodes. The random numbers are generated off-line at one node and distributed to all nodes involved in the communication.

Figure 5 illustrates the patterns when $P=4$.

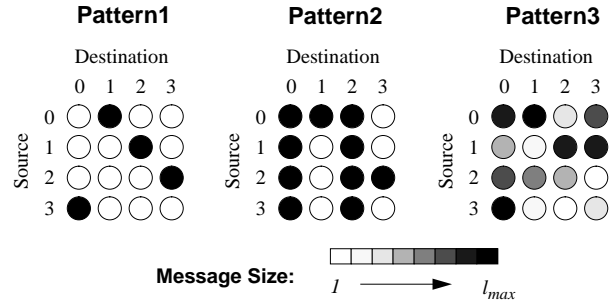


Figure 5. All-to-all communication patterns for the experiments

4.1. Node Contention

Table 5: Transmission latency on CM5 ($\mu\text{sec}/\text{token}$), when $P=64$

Pattern	Pattern1			Pattern2			Pattern3		
	l_{max}	128	256	512	128	256	512	128	256
A1	86.14	76.66	81.57	70.58	95.93	71.28	20.79	18.58	23.32
A2	14.01	15.02	20.32	33.94	30.09	53.14	20.72	10.12	14.55
A3	18.97	20.17	21.53	50.10	52.98	52.90	10.38	10.12	9.64
A4	13.74	14.82	15.37	25.06	29.18	16.30	13.19	9.59	13.43

Table 6: Transmission latency on T3D ($\mu\text{sec}/\text{token}$), when $P=64$

Pattern	Pattern1			Pattern2			Pattern3		
	l_{max}	128	256	512	128	256	512	128	256
A1	21.1	34.0	31.6	17.5	10.7	11.4	3.5	3.7	4.0
A2	18.4	11.7	9	7.1	10.0	9.0	3.9	4.9	5.7
A3	9.0	14.1	12.9	17.2	20.3	21.0	3.5	4.7	4.5
A4	0.9	6.0	5.4	5.6	6.5	6.8	4.5	5.4	5.9

We first focus on the node contention problem on the target machines. We measured the communication time for transmitting the tokens. This time was divided by the total number of tokens transmitted to obtain the average transmission latency. Note that the time does not include start-up times and local memory access times. Table 5 and Table

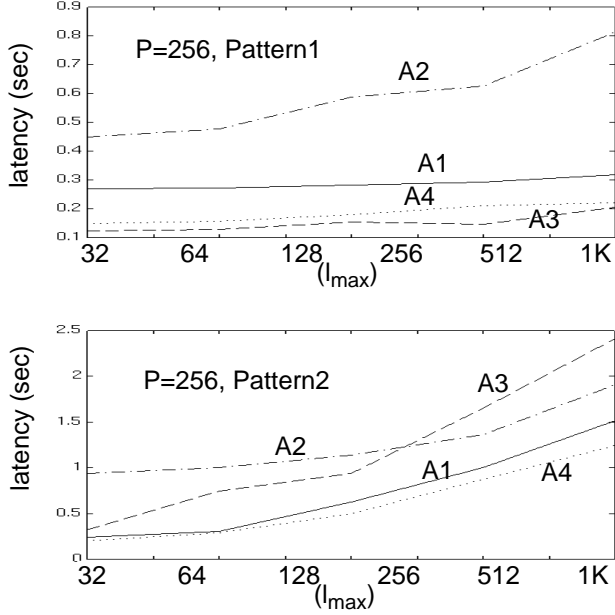


Figure 6. Experimental results on CM5

6 list the results on CM5 and T3D. The results on SP2 are not listed since we could not execute our code in dedicated mode on SP2 at the time of this writing. The interference from other tasks can cause node contention. Our observations from the results are:

- On CM5, in the cases of Pattern1 and Pattern2, the transmission latencies are 4 to 6 times larger when A1 or A3 is used compared with the case when A2 or A4 is used.
- On T3D, in the cases of Pattern1 and Pattern2, the transmission latencies are 3 to 7 times larger when A1 or A3 is used compared with the case when A2 or A4 is used.
- When A2 or A4 is used, the transmission latency does not vary significantly on different communication patterns.

4.2. Message Passing Latencies

The results of the case studies are shown below:

4.2.1 Case study 1: CM5

Figure 6 shows the message passing latencies (the total time to perform the communication operation) in the case of CM5. When $P = 256$, A4 has the least latency when Pattern2 is used. Also, as the message size increases, the latency grows faster in the case of A3 than in the case of A4.

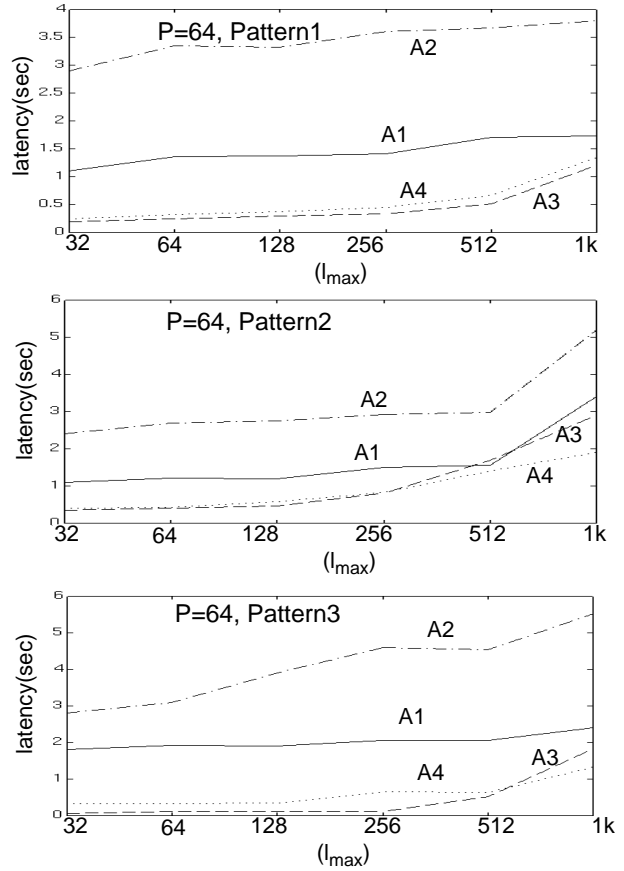


Figure 7. Experimental results on SP2

4.2.2 Case study 2: SP2

Figure 7 shows the results of our experiments on SP2. A3 and A4 are superior to A1 and A2 in all of the cases considered. A4 is a superior algorithm when l_{max} is greater than 256 in the case of Pattern2, greater than 512 in the case of Pattern3 and greater than 1K in the case of Pattern1.

4.2.3 Case study 3: T3D

Figure 8 shows the message passing latencies in the case of T3D. On Pattern1, when $P = 256$, A4 is attractive if l_{max} is greater than 512

4.3. Overlapping communication with computation

We modified algorithm A4 to overlap the communication with computation using nonblocking MPI primitives. By careful structuring of the computations, the local memory access and data transfer between the main memory and the interface buffer can be overlapped with communication using the nonblocking send/receive commands.

Figure 9 illustrates the communication latencies on CM5, T3D, and SP2 using blocking and nonblocking communication modes respectively. The latencies were measured using Pattern3. The results show that the use of nonblocking com-

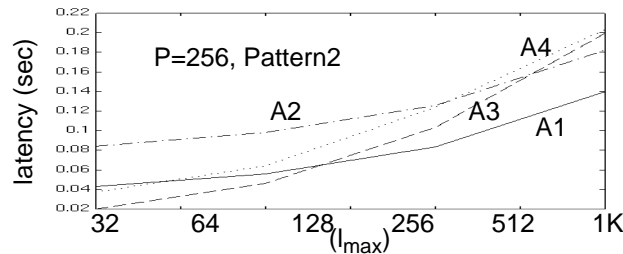
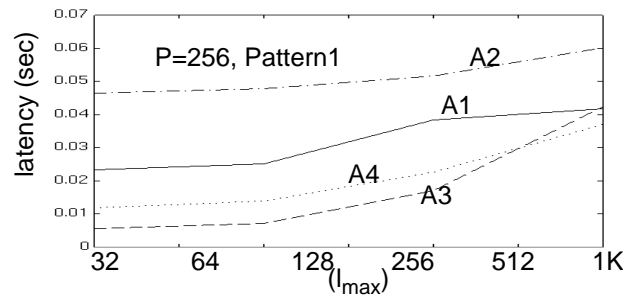
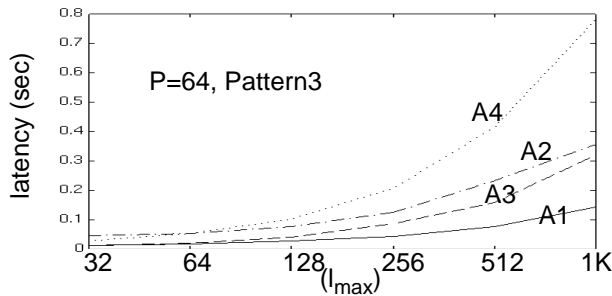
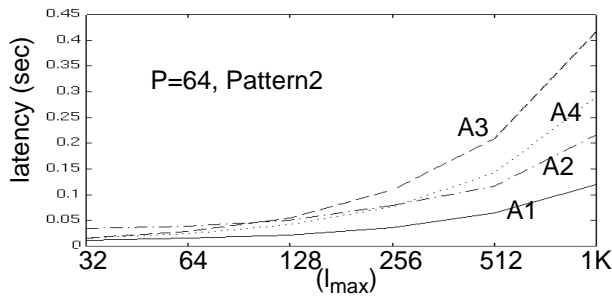
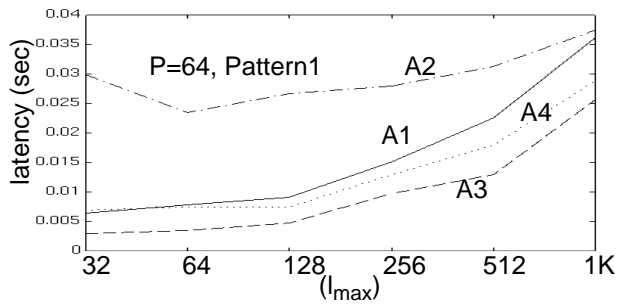


Figure 8. Experimental results on T3D

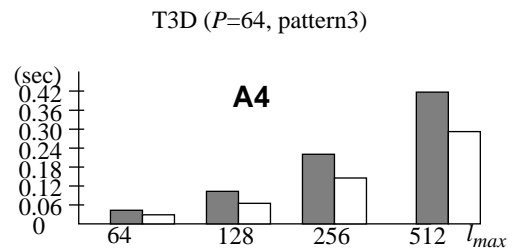
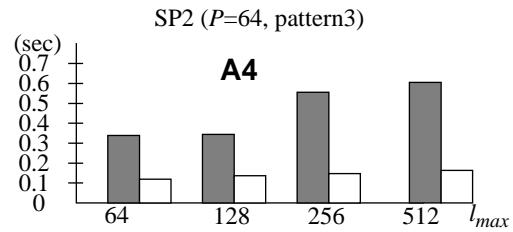
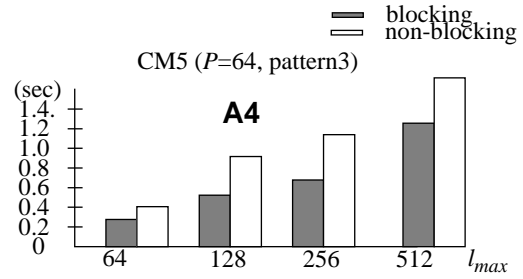


Figure 9. Overlapping communication with computation in A4 algorithm

mands results in improved performance on SP2 and T3D compared with using blocking communication because there are communication processors at the nodes to off-load communication related operations on these machines. On the other hand, the implementations using nonblocking communication performs worse on CM5. On CM5, the main processor at each node is busy during the entire message passing, and the non-blocking transmission mode results in overheads due to additional commands to test for message arrival.

5. Discussion and concluding remarks

Based on the architectural features of the machines, the message sizes, and the message irregularities, several trade-offs arise in selecting the appropriate solutions. The transmission rate is relatively higher when we use algorithm A2 or A4 than the case when algorithm A1 or A3 is used. A3 is less attractive when the outgoing traffic increases since the overheads due to memory accesses become dominant. As P increases, A3 and A4 perform better since the start-up latency grows as \sqrt{P} . Pattern1 and Pattern2 cause higher node contention than Pattern3.

Our algorithm is suitable for performing all-to-all com-

munication with high variance in message size, and the number of nodes is large.

The communication time is reduced by reducing the number of message start-ups and smoothing out the variance of the message size. On SP2, and T3D, the performance can be further improved by overlapping the communication with computation. The overhead due to local memory access adversely affects our algorithm and it masks the potential performance gains. Although we can employ nonblocking message passing primitives to reduce the latency on SP2 and T3D, there are still some operations (about 40 percent of the memory access latency on T3D) which should be executed in sequential order and cannot be overlapped with communication. To achieve further performance improvement, we can exploit lower-level (machine-dependent) features to reduce memory copy times. For example, message decomposing and coalescing can be performed directly between the local memory and the interface buffer in some HPC platforms.

Our implementations have been performed using the MPI primitives. Thus, the code is portable to other platforms. We employed MPI point-to-point communication primitives only. Therefore, the performance of the implementations is independent of the efficiency of the current MPI collective communication primitives. Our algorithm can be easily ported to other distributed systems such as NOWs. The communication latency becomes more significant in those systems due to limited bandwidth of such systems compared with general purpose HPC systems. In such a scenario, our algorithm is attractive for reducing the communication overheads.

6. Acknowledgment

The implementations were performed on SP-2 at the Maui High Performance Computing Center (MHPCC). This research is sponsored in part by the Phillips Laboratory, Air Force Material Command, USAF, under cooperative agreement number F29601-93-2-0001. Implementations on T3D were performed at the Pittsburgh Supercomputing Center (PSC).

References

- [1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team, "A Case for NOW (Networks of Workstations)," IEEE Micro., presented at Hot Interconnects II and at Principles of Distributed Computing, August 1994. URL: <http://now.cs.berkeley.edu/Case/case.html>
- [2] R. Ponnusamy, Y. Hwang, R. Das, and J. Saltz, "Supporting Irregular Distributions Using Data-Parallel Languages," IEEE Parallel and Distributed Technology, Vol. 3, No. 1, Pages 12-24, Spring 1995.
- [3] S. Hambrusch, F. Hameed and A. Khokhar, "Communication Operations on Coarse-Grained Mesh Architectures," Parallel Computing, Vol. 21, pp 731-751, 1995.
- [4] S. Ranka, R. Shankar and K. Alsabti, "Many-to-Many Communication With Bounded Traffic," 1995 Symposium on Frontiers of Massively Parallel Computation.
- [5] Myricom Corporation, "Myrinet User Manual," URL: <http://www.myri.com:80/documentation>
- [6] V. Karamcheti, and A. A. Chien, "Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D," Proc. of ISCA'95, June 1995.
- [7] C. Stunkel, D. G. Shea, et. al, "The SP2 Communication Subsystem," IBM Highly Parallel Supercomputing Systems Laboratory, August 22, 1994.
- [8] Y. Chang, N. Golmie, D. Su, "Study of Interoperability between EFCI and ER Switch Mechanism for ABR Traffic in an ATM Network," Proceedings of the Fourth International Conference on Computer Communications and Networks (ICCCN95), Sep. 1995.
- [9] H. Xu, and T. W. Fisher, "Improving PVM Performance Using ATOMIC User-Level Protocol," High-Speed Network Computing Workshop '95 at IPPS'95, April 1995.
- [10] T. Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," Proc. of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, Dec. 1995.
- [11] A. Choudhary, G. Fox, S. Hiranadani, K. Kennedy, C. Koebel, S. Ranka, J. Saltz, "Classification of Irregular Loosely Synchronous Problems and their Support in Scalable Parallel Software Systems," in 1992 DARPA Software Technology Conference Proceedings, pp138- 149, Syracuse Technical Report SCCS-255.
- [12] S. Ranka, J. Wang, and G. C. Fox, "Static and Runtime Algorithms for All-to-Many Personalized Communications on Permutation Networks," Proceedings of the 1992 International Conference on Parallel and Distributed Systems, pp. 211-218, HsinChu, Taiwan, Dec. 1992.
- [13] C. Wang, V. K. Prasanna, and Y. Lim, "Parallelization of Perceptual Grouping on Message-Passing Machines," 1995 Workshop on Computer Architecture for Machine Perception, Sep. 1995.
- [14] C. Wang, V. K. Prasanna, H. Kim, A. Khokhar, "Scalable Data Parallel Implementations of Object Recognition using Geometric Hashing," Journal of Parallel and Distributed Computing, pp. 96-109, March 1994.
- [15] C. Wang, and V. K. Prasanna, "Portable and Scalable Algorithms for Irregular All-to-all Communication", Manuscript of Department of EE-Systems, University of Southern California, April 1994.
- [16] J. Demmel and S. Smith, "Parallelizing a Global Atmospheric Chemical Tracer Model," Symposium on High Performance Computing and Communications, May 1994.