Scheduling Regular and Irregular Communication Patterns on the CM-5

Ravi Ponnusamy
Syracuse University, Northeast Parallel Architectures Center

Rajeev Thakur
Syracuse University, Northeast Parallel Architectures Center

Alok Choudhary
Syracuse University, Northeast Parallel Architectures Center

Geoffrey C. Fox
Syracuse University, Northeast Parallel Architectures Center

Scheduling Regular and Irregular Communication Patterns on the CM-5

Ravi Ponnusamy Rajeev Thakur Alok Choudhary Geoffrey Fox
Northeast Parallel Architectures Center, 1-019 CST, Syracuse Univ., Syracuse, NY 13244-4100

Abstract

In this paper, we study the communication characteristics of the CM-5 and the performance effects of scheduling regular and irregular communication patterns on the CM-5. We consider the scheduling of regular communication patterns such as complete exchange and broadcast. We have implemented four algorithms for complete exchange and studied their performances on a 2D FFT algorithm. We have also implemented four algorithms for scheduling irregular communication patterns and studied their performance on the communication patterns of several synthetic as well as real problems such as the conjugate gradient solver and the Euler solver.

1 Introduction

The performance of a distributed memory computer depends to a large extent on how fast interprocessor communication can be performed. Despite significant improvements in design, scalability and the underlying technology of parallel computers, the improvements in communication time have lagged far behind those in the computation power of each node. It is still two orders or more expensive to access a remote datum than to access a local datum.

This paper presents an experimental study of the communication capabilities of the Connection Machine 5 (CM-5) and the problem of scheduling regular and irregular communication patterns on the CM-5. Similar studies have been performed for other parallel machines such as Intel iPSC/2 [3], Symult 2010 [5], Intel iPSC/860 [1, 2] and CM-2 [15]. We have done this study taking into account the fact that the current version of CM-5 software supports only synchronous communication.

Section 2 presents some details of the CM-5 architecture. The problem of scheduling regular communication patterns is considered in section 3. We have implemented four algorithms for complete exchange and two for broadcast and studied their performance for various message and machine sizes. Section 4 presents four algorithms for scheduling irregular communication patterns. We have studied the performance of these problems on several synthetic as well as real problems such as the conjugate gradient solver and the Euler solver [12]. Finally, conclusions are presented in Section 5.

2 The CM-5 Architecture

The CM-5 is a scalable distributed memory multiprocessor system [6]. It can be scaled up to 16K processors. It supports both SIMD and MIMD programming models. Each node on the CM-5 is a SPARC processor which can operate at a peak speed of 32 MIPS and has four optional vector processors. Thus, each node is capable of peak 128 MFLOPS The nodes can be organized into a single partition or multiple partitions. Each partition has a manager which governs the allocation of parallel resources.

The CM-5 has two internal networks that support interprocessor communication - 1) control network and 2) data network. The control network supports operations that require global communication such as global reduction operations, parallel prefix operations and processor synchronization. It has a latency of about 2–5 microseconds. The data network supports point-to-point communication. The data network topology is a fat tree as shown in Figure 1.

Both data and control networks have a peak bandwidth of 20 MBytes/Sec. The maximum bandwidth is obtained when communication takes place among nodes in the same cluster of four processors. A data message is broken into a collection of packets. The packet size is 20 bytes, of which 16 bytes are for user data and the remaining 4 bytes contain control information such as destination and size. The CM-5 router employs a random routing scheme, and therefore, the packets may be received in random order. The data network guarantees a system-wide minimum bandwidth of 5 MBytes/sec no matter where the data is being sent in the system. The data network has a communication latency - sending a 0 byte message of 88 microseconds. We used CMMD library functions to do all our experiments. A detailed discussion of interprocessor communication overhead on the CM-5 can be found in [14, 4]. Further details of the CM-5 architecture can be found in [6].

Figure 1: CM-5 Data Network with 16 Processing Nodes

Table 1: 8 Processor Communication Schedule for Linear Exchange

Table 2: 8 Processor Communication Schedule for Pairwise Exchange

3 Scheduling Regular Communication Patterns

A regular communication pattern is one in which the pattern of data access is regular and can be detected at compile time; for example shift, complete exchange, broadcast etc.

The complete exchange (all-to-all personalized) communication pattern is commonly encountered in computations such as matrix transpose and two-dimensional FFT [2, 10]. Scheduling regular communication patterns on hypercubes can be done using CrOS III communication system described in [7]. In this section we study the behavior of four algorithms for complete exchange on the CM-5.

3.1 Linear Exchange (LEX)

This is the simplest of the four algorithms. For an N processor system, there are N steps in the algorithm. In step i, $0 \leq i < N$, processor i receives messages from every processor except itself. The entry $i \leftarrow j$ in table 1 indicates that processor i receives a message from processor j. The current version of CM-5 supports only synchronous communication. Since at each step all processors send messages to a particular processor i, synchronous communication will adversely affect the performance. If asynchronous (or non-blocking) communication is allowed, processors need not wait for their messages to be received in step i in order to proceed to step $i+1$.

3.2 Pairwise Exchange (PEX)

The Pairwise Exchange algorithm is shown in Figure 2. There are $N-1$ steps in an N processor system. The communication schedule for this algorithm is as follows. At step i, $1 \leq i \leq N-1$, each processor exchanges a message with another processor determined by taking the exclusive-or of its processor number with i. Therefore, this algorithm has the property that the entire communication pattern is decomposed into a sequence of pairwise exchanges. The communication schedule of the pairwise exchange algorithm for 8 processors is given in Table 2. The entry $i \leftrightarrow j$ in the table indicates that processors i and j exchange messages.

The PEX algorithm is better than LEX in terms of utilizing the bandwidth of the network and reducing

```
do j= 1, nproc - 1
  node = xor(mynumber, j)
  if (mynumber < node )
    receive(node)
    send(node)
  else
    send(node)
    receive(node)
  end if
end for
```

Figure 2: Pairwise Exchange Algorithm

Table 3: 8 Processor Communication Schedule for Recursive Exchange

Step 1	Step 2	Step 3
$0 \leftrightarrow 4$	$0 \leftrightarrow 2$	$0 \leftrightarrow 1$
$1 \leftrightarrow 5$	$1 \leftrightarrow 3$	$2 \leftrightarrow 3$
$3 \leftrightarrow 6$	$4 \leftrightarrow 6$	$4 \leftrightarrow 5$
$5 \leftrightarrow 7$	$5 \leftrightarrow 7$	$6 \leftrightarrow 7$

```
bytes = Size/2
for i = 0, $\lg N - 1$
  k = N/pow(2, i)
  if (mod(mynumber, k) < k/2)
    node = mynumber + k/2
  else
    node = mynumber - k/2
  if (mynumber < node )
    pack_message_to_send
    send (node)
    receive(node)
    unpack_received_message
  else
    receive(node)
    unpack_received_message
    pack_message_to_send
    send(node)
  end if
end for
```

Figure 3: Recursive Exchange Algorithm

processor idle time. This algorithm is known to perform well on Intel hypercubes and has been used in other studies such as in [2, 8, 16].

3.3 Recursive Exchange (REX)

The Recursive Exchange algorithm is a $\lg N$ step algorithm for a system with N processors. Each message is of size $n \times N/2$ for an exchange involving n bytes per processor. The algorithm is shown in Figure 3. The communication schedule of the REX algorithm for 8 processors is given in Table 3.

Although this algorithm takes less number of steps than the other two algorithms, the amount of data transmitted in each step is much higher. Since it is a store-and-forward algorithm, each step incurs additional overhead of reshuffling data [10].

3.4 Balanced Exchange (BEX)

In the pairwise exchange algorithm, the communication schedule is such that in the first four steps, all processors in a cluster of four processors communicate with each other. That is, in the first four steps, all the communication is between nearest neighbors. In the next four steps, all processors in a cluster of four communicate with processors in a neighboring cluster and so on. Since all four processors try to do so simultaneously, there is contention. Instead of having a communication schedule in which all processors first communicate within a cluster and then all communicate with some remote cluster, one can have a more balanced schedule in which at every step two processors in a cluster communicate with each other and two communicate with processors in a remote cluster. This balances the amount of local and remote communication, so that all processors do not try to simultaneously communicate over a long distance. We call this algorithm as *balanced exchange algorithm* (BEX). BEX algorithm is particularly suitable for CM-5 *fat-tree* architecture as contention at the root of the tree is reduced. Unlike pairwise exchange algorithm, in this algorithm messages passing through the root of the fat-tree are *optimally* distributed across each step in the algorithm.

Such a balanced exchange algorithm (BEX) can be obtained by a simple modification of the pairwise exchange algorithm as shown in Figure 4. For the purpose of determining the communicating pairs of processors, we define a mapping between the physical number of a processor and its virtual number as

virtual no. = physical no. - 1

If virtual no. = -1 then virtual no. = N - 1

where N is the total number of processors in the system.

With this mapping, if we apply the pairwise exchange algorithm using the virtual processor numbers, we get the communication schedule shown in Table 4 which is balanced with respect to local and remote communications. In an N ($N mod 16 = 0$) processor system, $3N/4 \times N/2$ exchange pairs (*global exchanges*) use the root of tree to perform complete exchange. The PEX algorithm schedules complete exchange in

```
virtual = (mynumber + 1) MOD nprocs
do j= 1, nprocs - 1
  node = xor(virtual, j) - 1
  if (node == -1)
    node = nprocs - 1
  end if
  if (mynumber < node )
    receive(node)
    send(node)
  else
    send(node)
    receive(node)
  end if
end for
```

Figure 4: Balanced Exchange Algorithm

Table 4: 8 Processor Communication Schedule for Balanced Exchange

$N-1$ steps such that $3N/4$ steps have all *global exchanges*. But, the BEX algorithm schedules complete exchange in $N-1$ steps such that the *global exchanges* are distributed across $N-1$ steps.

3.5 Performance of the Complete Exchange Algorithms

Figure 5 compares the communication time of the four exchange algorithms on a 32 node CM-5. The message size was varied between 0 and 2048 bytes. Due to the synchronous communication constraint, the LEX algorithm performs much worse than the other algorithms. Therefore we did not consider it for any further analysis. For small message sizes, the performance of PEX, REX and BEX is virtually indistinguishable on this scale. However, for large message sizes, PEX performs much better than REX and BEX performs better than PEX. This is because of the following two reasons. First, even though the number of steps in REX is only $\lg N$, as compared to N steps in PEX, the message size in REX remains constant at $n \times N/2$, whereas the size of each message in PEX is n. Second, each node needs to buffer and reshuffle data

Figure 5: Complete Exchange Algorithms on 32 nodes

Figure 6: Complete Exchange Algorithms on Varying Multiprocessor Sizes (message sizes = 0, 256 Bytes)

in REX so that appropriate data can be sent to the appropriate node. These two overheads outweigh the savings in the number of communication steps. BEX performs the best because it balances local and remote communication at each step.

We selected a few message sizes in different ranges, and collected the communication times for several machine sizes. Figures 6, 7 and 8 show the communication times on up to 256 processors for algorithms REX, PEX and BEX. Figure 6 shows times for messages of size 0 bytes and 256 bytes, Figure 7 shows times for messages of size 512 bytes and Figure 8 shows times for messages of size 1920 bytes.

Clearly for messages of size 0 byte, REX performs better than PEX and BEX for all multiprocessor sizes because there is no data shuffling involved and it has only $\lg N$ exchanges compared to $N-1$ exchanges in PEX and BEX. For messages of size 256 bytes, PEX performs better than REX for small multiprocessor

Figure 7: Complete Exchange Algorithms on Varying Multiprocessor Sizes (message size = 512 Bytes)

Figure 8: Complete Exchange Algorithms on Varying Multiprocessor Sizes (message size = 1920 Bytes)

sizes because the overhead of message size and number of steps dominate for REX. As the number of processors increases, the overhead of the larger number of messages dominates the overhead of larger message size and reshuffling in REX, and therefore, REX performs better. BEX performs the best for messages of size 256 bytes. For message sizes of 512 and 1920 bytes, and small multiprocessor sizes, BEX and PEX perform better than REX. But for large multiprocessor sizes, REX performs the best.

We implemented a 2D FFT algorithm using these complete exchange algorithms. The 2D array is distributed along rows among processors. Each Processor initially performs 1D FFT operation on its local data and performs a complete exchange using anyone of the algorithms described. Each processor then, performs 1D FFT on new data. The performance of this 2D FFT on various sizes of data are shown in table 5.

```
for j = 1, lg $N$
   distance = N/pow(2, j);
   if (mod(mynumber, distance) == 0) then
      if (mod(mynumber/distance, 2) == 0) then
         send(node);
      else
         receive(node);
      end if
   end if
end for
```

Figure 9: Recursive Broadcast Algorithm

3.6 Broadcast

Broadcast is a very common communication primitive encountered in many applications. We consider one-to-all broadcast (also known as single source broadcast) [11]. This section presents the performance of two broadcast algorithms; namely, Linear Broadcast (LIB) and Recursive Broadcast (REB). We compare these algorithms with the system broadcast function.

The LIB is the simplest broadcast algorithm. It has $N - 1$ steps. The processor broadcasting a message simply sends the message one by one to all the processors. In the REB algorithm, there are $\lg N$ steps. Without loss of generality, consider processor 0 to be the broadcasting source. In the first step, it sends the message to processor $N/2$, in the second step processor 0 sends the message to processor $N/4$ and processor $N/2$ sends the message to processor $3N/2$, and so on. The REB algorithm is given in Figure 9.

Figure 10 shows the performance of the two algorithms and the broadcast function provided by the system [4] as a function of message size for a 32 node machine partition. Clearly, the LIB algorithm performs much worse than the REB algorithm. Therefore, we did not consider the LIB algorithm any further. The REB performs better than the system broadcast when the message size is more than 1K byte. The REB selectively broadcast to a particular group of processors in a partition whereas, the current version of the system broadcast function requires all processors in the partition to participate in the process. Selective broadcasting is sometimes necessary for instance, when processors are configured as a mesh and broadcast along a row or a column is required.

Figure 11 shows the performance of the REB algorithm and the system broadcast as a function of multiprocessor size for various message sizes. The performance of the built-in broadcast was almost the same irrespective of the number of processors in the system. So, we have shown only one curve for it in the

Table 5: Performance of Scheduling Algorithms on 2D FFT (Time in Secs.)

	No. Procs = 32				No. Procs = 256			
	Scheduling Algorithm				Scheduling Algorithm			
Array Size	Linear	Pairwise	Recursive	Balanced	Linear	Pairwise	Recursive	Balanced
256x256	0.215	0.152	0.112	0.114	4.340	0.076	0.077	0.076
512x512	0.845	0.470	0.467	0.470	4.750	0.120	0.120	0.120
1024x1024	3.135	2.007	2.480	2.005	5.968	0.314	0.313	0.312
2048x2048	14.780	9.032	9.245	8.509	18.087	1.738	2.160	1.668

Figure 10: Broadcast Algorithms on 32 nodes

Figure 11: Recursive Broadcast Algorithm on Varying Sizes of Nodes

figure. For small size messages, the system broadcast function performs better than the REB. However, as the message size, the REB is better than the system broadcast. For instance, the REB is better than the system when the message size is more than 2K bytes when the number of processors is 256.

4 Scheduling Irregular Communication Patterns

An irregular problem is one in which the pattern of data access is input-dependent [13, 7]. Hence, when an irregular problem is implemented on message passing machines, the communication between the processors will also be irregular and will not be known beforehand. Such irregular communication patterns occur in a large number of computationally intensive problems such as unstructured mesh methods used to solve problems in computational fluid dynamics. To optimize communication between processors, the communication patterns in these problems can be captured and scheduled at runtime. Such dynamic scheduling of messages on hypercube can be done by using *crystal_router* described in [7]. The performance effects of irregular communication patterns on the CM-2 have been studied in [15]. In this section we study their effects on the CM-5.

We have implemented four different algorithms for scheduling irregular communication patterns namely Linear Scheduling (LS), Pairwise Scheduling (PS), Balanced Scheduling (BS) and Greedy Scheduling (GS). We have studied the performance of these algorithms for communication patterns of synthetic as well as real problems such as conjugate gradient solver and Euler solver for several data sets. A communication pattern is represented as a two-dimensional array called 'Pattern'. The element Pattern[i][j] indicates the number of bytes to be sent from processor i to processor j.

4.1 Linear Scheduling (LS)

Linear Scheduling is a modification of the linear exchange algorithm discussed previously to include the fact that the communication is irregular. At every step, each processor checks the communication matrix to see whether the operation to be performed is either an exchange, send, receive or no communication at all. If the matrix indicates no communication, the processor remains idle in that step. An example irregular communication pattern 'P' for 8 processors is given in Table 6. The communication schedule of the

Table 6: An Irregular Communication Pattern 'P'

0	1	0	1	0	1	1	0
1	0	1	0	1	1	1	1
0	1	0	1	0	0	0	0
1	0	1	0	1	1	1	0
0	1	1	1	0	1	0	1
0	1	0	0	1	0	1	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	1	0

Table 7: Communication Schedule for Pattern 'P' using Linear Scheduling

Table 8: Communication Schedule for Pattern 'P' using Pairwise Scheduling

Table 9: 8 Processor Communication Schedule for Balanced Exchange

linear scheduling algorithm for 8 processors with communication pattern 'P' is given in Table 7. The entire communication schedule is completed in 8 steps.

4.2 Pairwise Scheduling (PS)

The Pairwise Scheduling algorithm is a modification of the Pairwise Exchange algorithm of Figure 2 to take into account the irregular communication. The communicating pairs are determined in the same way as in pairwise exchange. But, in addition, each processor checks the communication matrix to see whether the operation to be performed is either an exchange, send, receive or no communication at all. If the matrix indicates no communication, the processor remains idle in that step. The communication schedule of the pairwise scheduling algorithm for 8 processors with communication pattern 'P' is given in Table 8. The entire communication is done in 6 steps.

4.3 Balanced Scheduling (BS)

The Balanced Scheduling algorithm is a modification of the balanced exchange algorithm given in Figure 4. The communicating pairs are determined in the same way as in balanced exchange. But, in addition, each processor checks the communication matrix to see whether the operation to be performed is either an exchange, send, receive or no communication at all. If the matrix indicates no communication, the processor remains idle in that step. The communication schedule of the balanced scheduling algorithm for 8 processors with communication pattern 'P' is given in Table 9. The entire communication is done in 7 steps.

4.4 Greedy Scheduling (GS)

In this algorithm each processor first uses a greedy strategy to determine the processors it has to communicate with at every step, and then uses this schedule to perform the communication. For a complete exchange operation this algorithm creates the same communication schedule as pairwise exchange. But when the communication is irregular, the greedy algorithm creates a different communication schedule than that by the pairwise scheduling algorithm. This is because in the greedy algorithm, if processor i does not have to communicate with processor j, it will communicate with the next available processor with which it needs to communicate. In the pairwise scheduling algorithm, if a pair of processors $[i, j]$ determined by the algorithm do not have to communicate, they remain idle in that step. The communication schedule of the greedy scheduling algorithm for 8 processors with communication pattern 'P' is given in Table 10. The entire communication is done in 6 steps.

4.5 Performance Comparison

The communication schedule needs to be created only once and can be used thereafter to perform the communication for as many iterations as required. Hence the time to compute the schedule can be amortized over all the iterations. We have created synthetic communication patterns with different communication

```
while ($msgs\_to\_send$ != 0) do
  iteration = iteration + 1
  for i = 1 to nprocs do
    $P_i$ selects the next available $P_j$
        among the processors it has to send to
    If $P_j$ also sends to $P_i$ then do an exchange
    Mark $P_i$ and $P_j$ as unavailable for this iter
    Decrement $msgs\_to\_send$ appropriately
  end for
end while
```

Figure 12: Greedy Scheduling Algorithm

Table 10: Communication Schedule for Pattern 'P' using Greedy Scheduling

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
$0 \leftrightarrow 1$	$0 \leftrightarrow 3$	$0 \rightarrow 5$	$0 \leftrightarrow 6$	$1 \rightarrow 6$	$1 \leftrightarrow 7$
$2 \leftrightarrow 3$	$1 \leftrightarrow 2$	$1 \leftrightarrow 4$	$1 \leftrightarrow 5$	$3 \rightarrow 5$	$6 \rightarrow 2$
$4 \leftrightarrow 5$	$4 \leftrightarrow 7$	$3 \leftrightarrow 6$	$3 \leftrightarrow 4$	$2 \leftarrow 4$	
$6 \leftrightarrow 7$	$5 \leftrightarrow 6$			$7 \rightarrow 0$	

densities of 10%, 25%, 50% and 75% of complete exchange and studied the performance of the above algorithms on these patterns for message sizes of 256 and 512 bytes on a 32 processor system. The results are given in Table 11. We see that the linear scheduling algorithm performs the worst in all cases because of the synchronous communication constraint. The performance of the pairwise and balanced scheduling algorithms is comparable. The greedy algorithm performs the best for communication densities of less than 50%, because the number of steps involved in the communication is the minimum of all the algorithms. But when the communication density is higher than 50%, the greedy algorithm may require more number of steps than the pairwise and balanced algorithms, which degrades the performance. In this case, balanced scheduling performs the best.

The performance of these algorithms on real problems such as the conjugate gradient solver and Euler solver for unstructured meshes of different sizes, is given in Table 12. The table shows the communication time for each algorithm as well as the average number of bytes transferred in each problem and the percentage of communication operations with respect to complete exchange. The communication percentage varies from 9% in the conjugate solver to 44% in the Euler solver for meshes with 2K and 9K vertices. The average number of bytes transferred per communication operation varies from 85 bytes for the Euler solver for a mesh with 545 vertices to 643 bytes for the conjugate gradient solver. The performance of the algorithms on the real problems is consistent with that on the synthetic patterns. Since the communication density is less than 50% in the real problems, the greedy algorithm performs the best.

5 Conclusions

This paper presented experimental results for communication overhead on the CM-5 and the performance effects of scheduling regular and irregular communication patterns on the CM-5. We studied the communication overhead of four complete exchange algorithms. For a large number of processors, the Recursive Exchange algorithm performs the best. Balanced exchange performs the best for small message sizes. For large message sizes in a small multiprocessor system, pairwise exchange performs better than the other algorithms.

We implemented two algorithms for one-to-all selective broadcast; namely, Linear Broadcast and Recursive Broadcast. The recursive broadcast algorithm performs better than linear broadcast and it is also better the system broadcast functions when the message size is large.

For irregular communication patterns, the greedy algorithm performs the best when the communication density is less than 50%. The balanced exchange algorithm performs the best when the communication density is higher than 50%. The linear scheduling algorithm suffers because of the synchronous communication constraint.

Acknowledgments

The authors would like to thank Joel Saltz for fruitful discussions about irregular communication scheduling. This work was sponsored in part by DARPA under contract no. DABT63-91-C-0028 and in part by NSF grant MIP-9110810.

References

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

[2] Bokhari,S.H., *Complete Exchange on the iPSC*, ICASE Technical Report 91-4, 1991

[3] Bomans L. and Roose D., *Benchmarking the iPSC/2 hypercube*, Concurrency: Practice and Experience, 1:3-18, 1989.

[4] Bozkus, Z., Ranka, S., and Fox, G., *Modelling the CM-5 Multicomputer*, in proceedings of Frontiers '92, October 92.

[5] Chittor S. and Enbody R., *Performance analysis of Symult 2010's interprocessor communication*

Table 11: Performance of Scheduling Algorithms for Synthetic Irregular Patterns on 32 Processors

Algorithms	Time (ms.)							
	10% Pattern		25% Pattern		50% Pattern		75% Pattern	
	256 bytes	512 bytes	256 bytes	512 bytes	256 bytes	512 bytes	256 bytes	512 bytes
Linear	4.723	6.116	11.67	15.34	29.01	38.27	50.14	66.63
Pairwise	1.766	2.275	3.977	5.193	6.324	8.360	7.882	10.52
Balanced	1.933	2.494	3.724	4.861	6.034	8.013	7.856	10.50
Greedy	1.597	2.044	3.266	4.192	6.009	7.934	9.241	12.29

Table 12: Performance of Scheduling Algorithms for Real Irregular Patterns on 32 Processors

	Time (ms.)				
Algorithms	Conj. Grad. 16K 9%, 643 bytes	Euler 545 37%, 85 bytes	Euler 2K 44%, 226 bytes	Euler 3K 29%, 612 bytes	Euler 9K 44%, 505 bytes
Linear	8.046	25.87	48.88	50.78	77.13
Pairwise	6.623	7.374	15.04	19.98	21.91
Balanced	7.188	7.386	15.07	17.57	20.19
Greedy	5.799	5.656	12.30	14.34	17.01

network, TR CPD-89-19, Michigan State University, CS, 1989.

[6] *CM-5 Technical Summary*, Thinking Machines Corp., Cambridge, MA., 1991.

[7] Fox, G. et al., *Solving problems on concurrent Processors Vol I*, Printice Hall, 1988.

[8] Furmanski, W., and Fox. G., *Hypercube Communication for neural network algorithms,* Caltech report C^3P-405, Feb 1987.

[9] Lee, M., Seidal, S.R., *Concurrent communication on the Intel iPSC/2*, Technical Report CS-TR 9003, Dept. of Computer Science, Michigan Tech. Univ., April 1990.

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

[11] Lennart Johnsson S. and Ho C. T., *Optimum broadcasting and personalized communication in hypercubes*, IEEE Trans. Computers, C-38(9):1249-1268, Sept., 1989.

[12] Mavriplis, D., *Three dimensional unstructured multigrid for the Euler equations*, In AIAA 10th Computational Fluid Dynamics Conference, June 1991.

[13] Ponnusamy, R., Saltz, J., Das, R., Koelbel, C., and Choudhary, A., *A Runtime Data Mapping Scheme for Irregular Problems*, in Proc. Scalable High Performance Computing Conference, April 1992.

[14] Ponnusamy, R., Choudhary, A., and Fox, G., *Communication Overhead on CM-5 : An Experimental Performance Evaluation*, Technical Report SCCS-252, Syracuse Center for Computational Science, March 1991.

[15] Saltz, J., Petiton, S., Berryman, H., and Rifkin, A., *Performance Effects of Irregular Communication Patterns on Massively Parallel Multiprocessors*, Journal of Parallel and Distributed Computing, Oct. 1991, pp 202-212.

[16] Schmiermund, T., Seidal, S.R., *A communication model for the Intel iPSC/2*, Technical Report CS-TR 9002, Dept. of Computer Science, Michigan Tech. Univ., April 1990.

[17] Seidal, S.R., Lee, M., and Fotedar, S., *Concurrent bidirectional communication on the Intel iPSC/820 and iPSC/2*, Technical Report CS-TR 9006, Dept. of Computer Science, Michigan Tech. Univ., April 1990.