Experimental Performance Evaluation of the CM-5

Ravi Ponnusamy **Rajeev Thakur** **Alok Choudhary**
Kishore Velamakanni **Zeki Bozkus** **Geoffrey Fox**

Northeast Parallel Architectures Center
111 College Place, Rm. 3-201
Syracuse University
Syracuse, NY 13244-4100

Abstract

In this paper, we present an extensive experimental performance evaluation of the communication capabilities of the CM-5. We first study the communication characteristics such as startup time, sustainable bandwidth for simple messages as a function of message size and number of processors, and the effect of multiple messages and link contention on the communication time.

We study the effect of dense communication patterns such as complete exchange and propose four algorithms for scheduling a complete exchange operation. We also consider the scheduling of irregular communication patterns and present four algorithms for the same. We have tested these algorithms on many synthetic irregular communication patterns as well as those arising in real problems such as the conjugate gradient solver and the Euler solver. Finally, we study the performance and communication aspects of scientific applications such as two-dimensional FFT and Gaussian Elimination on the CM-5.

These results are based upon a beta version of the software and, consequently, are not necessarily representative of the performance of the full version of the software.

1 Introduction

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

This paper presents an extensive experimental study of the communication capabilities of Thinking Machines Corporation's Connection Machine 5 (CM-5). We study important communication characteristics of the CM-5 and the scheduling of regular and irregular communication patterns. Similar studies have been performed for other parallel machines such as Intel iPSC/2 [2], Intel iPSC/860 [1] and CM-2 [12]. We also study the performance and communication aspects of some scientific applications like two-dimensional FFT and Gaussian Elimination.

The architecture of the CM-5 is described in Section 2. Section 3 describes the experiments performed to study the communication characteristics of the CM-5 and their results. We study the effect of size, distance, multiple messages and contention on the communication time. We discuss four algorithms for scheduling a complete exchange operation in Section 4 and study their performance for various message sizes and number of processors. Section 5 presents four algorithms for scheduling irregular communication patterns. We study the performance of these algorithms on several synthetic communication patterns as well as those arising in real problems such as the conjugate gradient solver and the Euler solver [9]. Section 6 discusses the performance and communication issues of parallel algorithms for two-dimensional FFT and Gaussian Elimination on the CM-5. Finally, conclusions are presented in Section 7. All the experiments in this paper have been performed using the message passing library CMMD Version 3.0 Beta.

2 The CM-5 Architecture

The CM-5 is a scalable distributed memory multiprocessor system which can be scaled up to 16K processors [4]. It supports both SIMD and MIMD programming models. Each node

Figure 1: CM-5 fat tree, courtesy [6]

on the CM-5 is a SPARC processor and has four optional vector processors providing a peak floating point performance of 128 Mflops. The CM-5 has two internal networks that support interprocessor communication — the *Control Network* and the *Data Network*.

The control network is responsible for communication patterns in which many processors may be involved in the processing of each datum, such as global reduction operations, parallel prefix operations and processor synchronization. The data network supports point-to-point communication and has a fat tree topology as shown in Figures 1 and 2. It is actually a 4-ary fat tree or quad tree, where each node has four children. Each internal node of the fat tree is implemented as a set of switches. The number of switches per node depends on where it is in the tree. Nodes at level 1 have two switches. The number of switches per node doubles for each higher level till level 3, from where on it quadruples. Each level 1 or level 2 switch has two parents and four children; switches at higher levels have four parents and four children. The data network has a guaranteed system-wide bandwidth of 5 Mbytes/sec. The maximum bandwidth possible is 20 Mbytes/sec when communication takes place among nodes in the same cluster of four processors.

A message is divided into a group 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 routing algorithm for the data network compares the destination

Figure 2: CM-5 Data Network with 64 Processing Nodes, courtesy [6]

address with the source address to determine how far up the tree the message must travel. The message can then take any path up the tree. This allows the switches to perform load balancing on the fly. Once the message has reached the necessary height in the tree, it must follow a *particular* path down. Further details of the CM-5 architecture can be found in [4].

3 Communication Overhead on the CM-5

Important metrics to measure the communication capabilities of a distributed memory parallel computer include startup time, sustainable bandwidth with and without contention, effect of locality, and the time taken to perform communication intensive operations such as complete exchange.

3.1 Outline of Experiments

The following notation will be used in the rest of the paper. A set of k messages from multiple sources to multiple destinations is denoted as a set of tuples $\{(s_1, d_1), ..., (s_k, d_k)\}$, where (s_i, d_i) denotes the (source, destination) pair of the i^{th} message. The following is a brief description of the experiments performed.

1. In the first experiment, we study what is the maximum bandwidth that can be sustained for a single message traveling the shortest possible distance for message sizes up to 10 Kbytes. This is done by sending a single message (called "simple send") {(0,1)}.

2. In the second experiment, we study the impact of path length on the communication time of a simple send.

3. The third experiment presents the effects of sending a set of two messages, called Double Send (DS). The motivation for this experiment is to study if random routing and alternate paths to the first level switching node are effectively utilized.

4. The fourth experiment studies the effect of contention when maximum number of messages are sent from one cluster to another. Here, each processor sends a message to a distinct processor in the destination cluster.

An average of 100 repetitions of each experiment were performed to determine the communication time accurately. The precision of the CM5 clock is 1 microsecond. We determine the communication time for a send as half the time for a round-trip message (the same technique has been used in [1]).

3.2 Message Size

In the first experiment, we study the communication time for sending a single message to another node in the same cluster of 4 processors (message set = {(0,1)}), for different message sizes. This represents the shortest possible distance a message would travel. Figure 3 shows the communication times for messages of size 0–100 bytes.

We define the startup time as the time taken for sending a message of size 0 bytes, which is observed to be 64 microseconds. An interesting observation from the figure is that the communication time for messages that are a multiple of 16 bytes is different from that for messages that are not a multiple of 16 bytes. Messages that are a multiple of 16 bytes take much less time than others, as indicated by the dips in curve of Figure 3. As stated earlier, a message in the CM-5 is sent as a sequence of packets. Each packet is 20 bytes long, of which 4 bytes are for control purposes and 16 bytes represent user data. If a message packet is full, i.e. if it contains 16 bytes of user data, the overhead of processing it is smaller than the

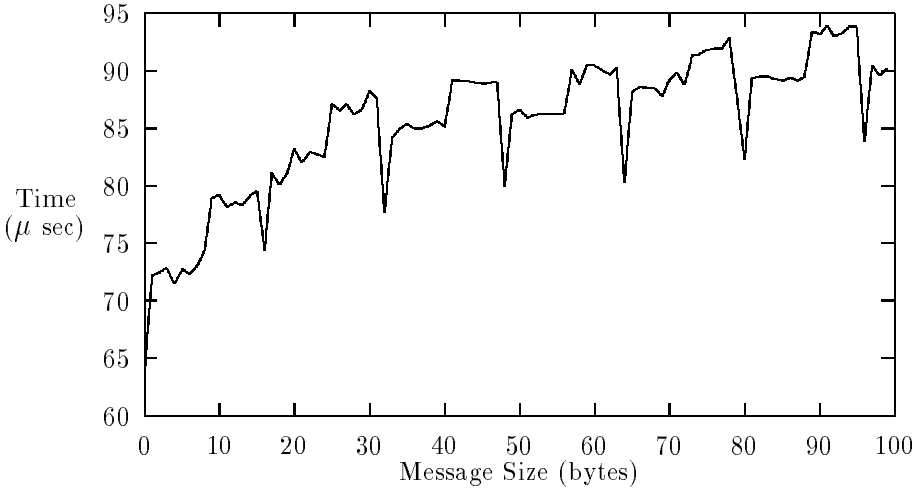

Figure 3: Message Size Varied from 0 to 100 bytes

overhead of processing a message otherwise. Hence, the communication overhead incurred will be smaller if the user made the message size a multiple of 16 bytes (by padding it if required). The above behavior is also observed when the message size is varied between 0 and 10,000 bytes as shown in Figure 4.

Using a linear chi-square fit, we obtain the communication time as a function of message size for communication within a cluster of 4 processors without contention as

$$t_{comm} = 0.112 \times l + 73 \ \ (in\ \mu secs.)$$

where, l is number of bytes and $l \, mod \, 16 = 0$; and

$$t_{comm} = 0.112 \times l + 83 \ \ (in\ \mu secs.).$$

where, $l \, mod 16 \neq 0$.

The above equations give a different time for a message of size 0 bytes than the observed time of 64 microseconds, because of the non-linearity in the graph in Figure 3 for message size 0 — 10 bytes.

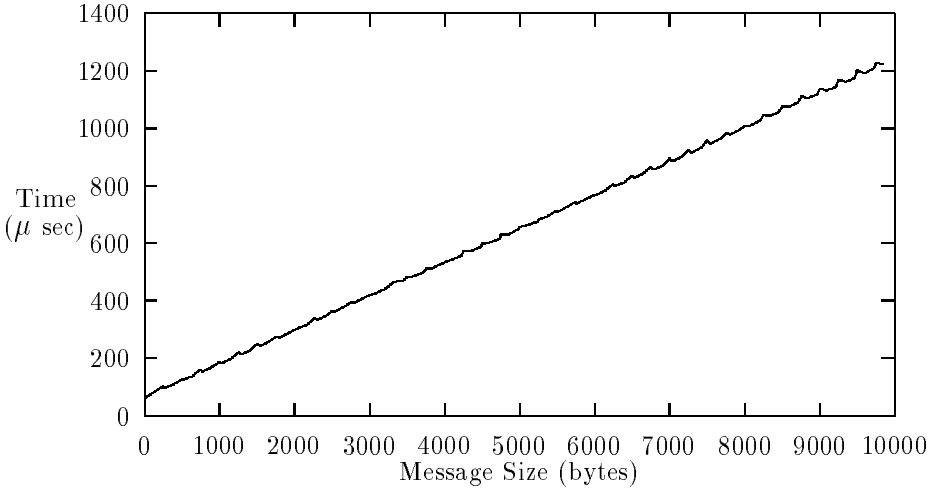

Figure 4: Message Size Varied from 0 to 10000 bytes

3.3 Impact of Path Length

Within a cluster of 4 processors, a message will traverse two links, one link up to the switching node and one link down to the destination processor. We call this two link traversal as path length of one because link traversals will always be a multiple of two (from the source, up to the least common ancestor, and down to the destination). Figure 5 shows the effect of distance on the communication time in a 512-node CM-5. The X-axis values 1, 2, 3, 4 and 5 correspond to a message sent by processor 0 to a destination processor in a cluster of 4, 16, 64, 256 or 1024 processors. For the sake of clarity we have shown graphs for four message sizes, 16, 512, 4K and 8K bytes; a representative message size from each range used in the previous experiments. In this experiment, only one pair of processors communicate. We observe that all four plots are perfectly horizontal, from which we conclude that the communication time is not affected by interprocessor distance when there is no contention.

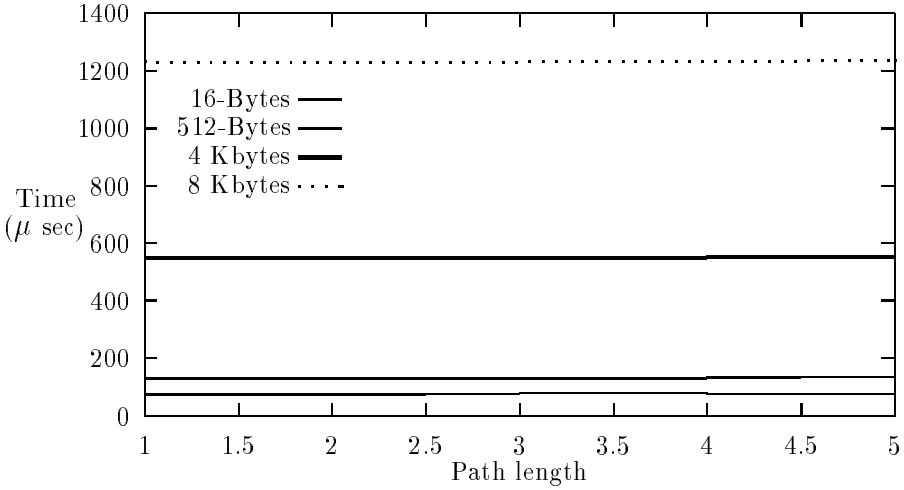

Figure 5: Effect of Path Length for Various Message Sizes

3.4 Multiple Messages and Contention

In this set of experiments we compare communication times for two distinct sources to two distinct destinations within and outside a cluster of four processors. The motivation behind these experiments is to check if the switching node which is the parent of nodes 0 through 3, can route two sets of messages with the same speed. Note that the CM-5 provides two paths for communication within a cluster of four processors, and it employs randomized routing. Therefore, it is expected that this experiment gives the same performance as the simple send experiments in the previous subsection.

Figure 6 compares a simple send (SS) {(0,1)} with a set of two messages (DS) {(0,1),(2,3)}. We observe that there is no significant difference in the performance of SS and DS. Hence, both parent nodes of a cluster are effectively used in this type of communication. Similar results are obtained when simple send (SS) is compared with double send (DS) for communication between processors that are four hops away (message set = {(0,4),(1,5)}), and that are six hops away (message set = {(0,16),(1,17)}).

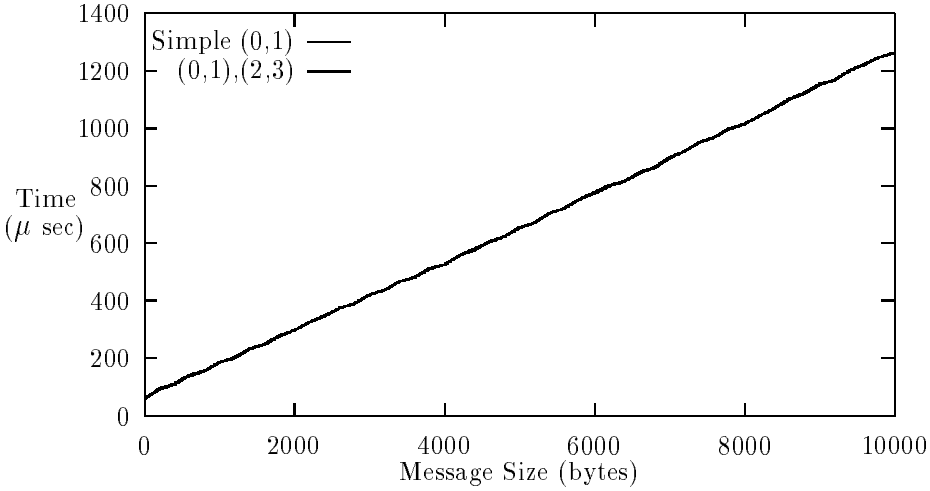

Figure 6: Comparing a simple send with double sends in a group of 4 processors

3.5 Maximum Number of Messages and Contention

In the following set of experiments, each node in a cluster communicates with a distinct node in the next cluster. Since all the processors try to communicate simultaneously, there is a possibility of contention. In the first experiment, shown in Figure 7, each processor in the cluster of processors 0 through 3 simultaneously and repeatedly communicates with nodes 4 through 7 in the neighboring cluster — the message set is {(0,4),...,(3,7)}. Compared to a simple send of one message, there is no appreciable difference in the communication times for messages of size up to 10 Kbytes. Hence, the presence of possible contention in small clusters does not affect the communication time much. This is because there are enough links for all processors in a cluster of four processors to simultaneously communicate with processors in an adjacent cluster of four, without contention.

When the cluster size is increased, the effect of contention becomes visible. In the next experiment, each node of a cluster of size 16 (nodes 0 through 15) communicates with the corresponding node of the next cluster (nodes 16 through 31) simultaneously and repeatedly. In this case the message set is {(0,16), (1,17), ..., (15,31)}. Figure 8 compares the

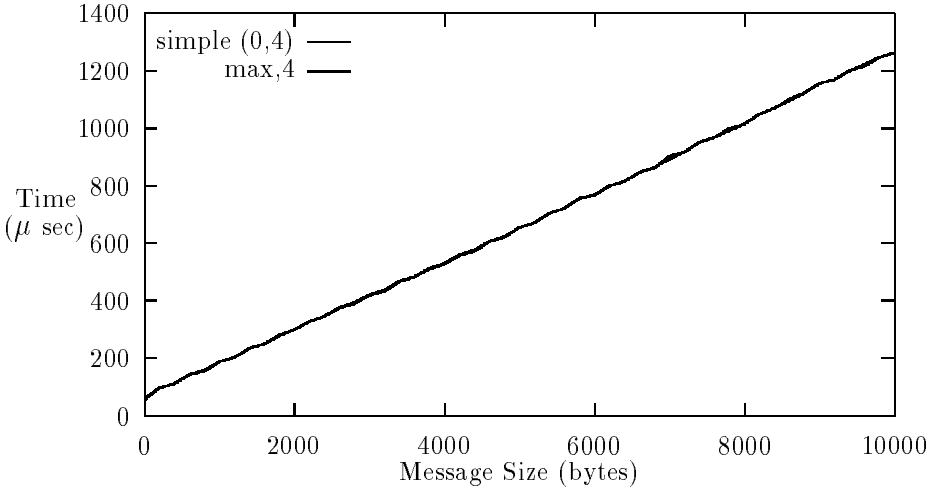

Figure 7: Comparing a simple send with max. no. of messages in a group of 8 procs.

communication times for the above message set with that of a simple send from processor 0 to processor 16. The communication time in the presence of contention is much greater than that for the simple send. The difference in the time also increases with increase in message size. This is because when a processor needs to communicate with a processor in a different cluster of 16 processors, messages have to travel 3 levels up the tree. There are only 8 links from a level 2 node to a level 3 node as can be observed from Figures 1 and 2. When 16 processors simultaneously try to access 8 links, there is contention which increases the communication time. A similar bottleneck exists at higher levels in the tree, so similar results are obtained when the number of processors is increased. Figure 9 shows the results on a 256 processor system.

4 Complete Exchange

The complete exchange or all-to-all personalized communication pattern arises very often in many important applications on distributed memory computers [1, 7]. Parallel quicksort, some implementations of the 2D FFT, matrix transpose, array redistribution etc. require

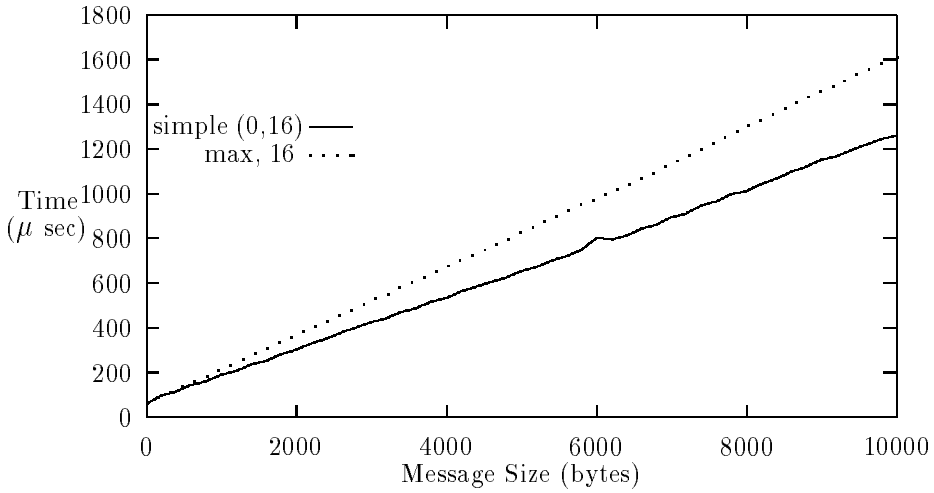

Figure 8: Comparing a simple send with max. no. of messages in a 32 proc. system

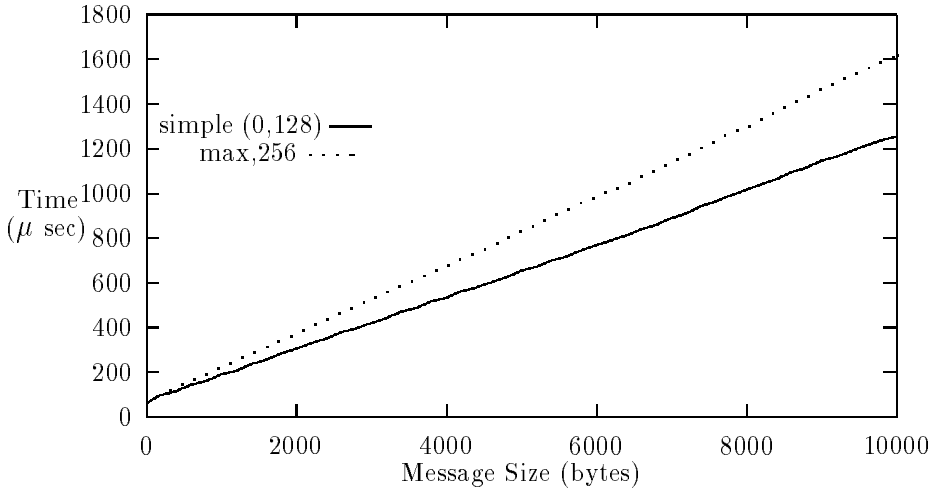

Figure 9: Comparing a simple send with max. no. of messages in a 256 proc. system

complete exchange. This is a communication pattern in which all processors simultaneously need to exchange data with all other processors. This is clearly the densest form of communication possible and it stretches the communication capabilities of the interconnection network to the limit. Hence, it is very important to use efficient algorithms to schedule a complete exchange operation, in order to get good performance. In this section we study the behavior of four algorithms for complete exchange on the CM-5 namely, Linear Exchange (LEX), Pairwise Exchange (PEX), Recursive Exchange (REX) and Balanced Exchange (BEX).

Each algorithm consists of several steps in which pairs of processors exchange with each other. The time for an exchange operation at step i can be written as

$$T = \alpha + L\ max(\beta_{ex}, f(i)\beta_{sat})$$

where

α = startup time

L = message size in each exchange operation

β_s = transfer time per byte for a single send with no link conflicts

β_{ex} = transfer time per byte for an exchange with no link conflicts

β_{sat} = transfer time per byte on a saturated link

$f(i)$ = maximum number of messages contending for a link at step i

We assume that conflicting messages share the bandwidth of a network link. The network may have excess bandwidth, enabling multiple messages to traverse a link in the same direction without conflict. In other words, $\beta_{sat} < \beta_{ex}$. The number of processors is denoted by p.

4.1 Linear Exchange (LEX)

The Linear Exchange algorithm is the simplest of the four algorithms. In step i, $0 \leq i < p$, processor i receives messages from every processor except itself. The algorithm clearly requires p steps. Since at every step one processor receives from all other processors, there is a lot of node and link contention. At step i, every processor sends data to processor i.

Table 1: Communication Schedule for PEX on 8 processors

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

Processor i has two links to its parent node and $p-1$ processors simultaneously need to use these links. Hence, the maximum number of messages contending for a link at any step is $\lceil \frac{p-1}{2} \rceil$. The time taken for any step i is given by

$$T(i) = \alpha + L\, max(\beta_s, \lceil \frac{p-1}{2} \rceil \beta_{sat})$$

The cost of LEX is obtained by summing over all steps of the algorithm :

$$T_{LEX} = \sum_{i=1}^{p-1} [\alpha + L\, max(\beta_s, \lceil \frac{p-1}{2} \rceil \beta_{sat})] = \alpha(p-1) + L(p-1) max(\beta_s, \lceil \frac{p-1}{2} \rceil \beta_{sat})$$

4.2 Pairwise Exchange (PEX)

The communication schedule for this algorithm is as follows. At step i, $1 \leq i \leq p-1$, 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. There are $p-1$ steps in a p processor system. The communication schedule of the pairwise exchange algorithm for 8 processors is given in Table 1. The entry $i \leftrightarrow j$ in the table indicates that processors i and j exchange messages. This algorithm is known to perform well on hypercubes and has been used in other studies such as in [1, 6, 13].

On the CM-5, when a processor has to communicate with another processor in its cluster of 4^k processors, $k \geq 1$, the message has to travel a maximum of k levels up the tree. The PEX algorithm has the property that in the first 15 steps, processors in a cluster of 16 processors exchange completely with each other and from step 16 onwards, they exchange with processors in other clusters. When a cluster of 16 processors exchange among themselves,

the messages have to travel either 1 or 2 levels up the tree depending on the source and destination. There are 32 links from level 0 to level 1 and 16 links from level 1 to 2 for this cluster, enough for the 16 processors to exchange among themselves without contention. However, when processors in a cluster of 16 need to exchange with processors in another cluster of 16, there are only 8 links from a level 2 node to a level 3 node, which results in 16 processors contending for 8 links, as explained in Section 3. A similar bottleneck exists at higher levels. For example, a level 3 node has 32 links upwards and downwards, and 64 processors in its subtree. Hence, in the first 15 steps of PEX there is no link contention. From step 16 onwards, the maximum number of messages contending for a link is 2 on an average. The time taken for step i is given by

$$T(i) = \begin{cases} \alpha + L\,\beta_{ex} & \text{for } 1 \leq i \leq 15 \\ \alpha + L\,max(\beta_{ex}, 2\beta_{sat}) & \text{for } i > 15 \end{cases}$$

The time for the entire PEX algorithm can be obtained by summing over all steps :

$$T_{PEX} = \sum_{i=1}^{15}[\alpha + L\,\beta_{ex}] + \sum_{i=16}^{p-1}[\alpha + L\,max(\beta_{ex}, 2\beta_{sat})]$$

which can be simplified to

$$T_{PEX} = (p-1)\alpha + L\,[15\beta_{ex} + (p-16)\,max(\beta_{ex}, 2\beta_{sat})]$$

For a complete exchange on 16 processors, this algorithm has no contention.

4.3 Recursive Exchange (REX)

In the Recursive Exchange algorithm, the number of processors is halved in each step and messages are exchanged over each cut. A processor sends all the data intended for all processors in another partition to only one processor in that partition, which forwards that data to the remaining processors in a later step. The number of steps required is $\lg p$ and each message is of size $L \times p/2$. The communication schedule for REX on 8 processors is given in Table 2. Although this algorithm takes less number of steps than LEX and PEX, the amount of data exchanged in each step is much higher. Since it is a store-and-forward type algorithm, each step incurs additional overhead of reshuffling data [7].

Table 2: Communication Schedule for REX on 8 Processors

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

In step i, $1 \leq i \leq \lg p$ each processor j exchanges with processor $j \pm \frac{p}{2^i}$. Communication always takes place either entirely within a cluster of 16 processors or entirely across clusters. In steps 1 to $\lg p - 4$, communication takes place across clusters, so the maximum number of messages contending for a link is 2. In steps $\lg p - 3$ to $\lg p$, communication takes place within a cluster of 16 processors, so there is no contention. Hence, the time taken by REX is

$$T_{REX} = \sum_{i=1}^{\lg p - 4} [\alpha + L\frac{p}{2} max(\beta_{ex}, 2\beta_{sat})] + \sum_{i=\lg p - 3}^{\lg p} [\alpha + L\frac{p}{2}\beta_{ex}]$$

which can be simplified to

$$T_{REX} = \alpha \lg p + L\frac{p}{2}[4\beta_{ex} + (\lg p - 4) max(\beta_{ex}, 2\beta_{sat})]$$

4.4 Balanced Exchange (BEX)

In the PEX algorithm, the communication schedule is such that all processors in a cluster first exchange completely with each other and then exchange with processors in other clusters. In other words, all the communication is either entirely within the cluster or entirely across clusters. As explained above, this gives rise to contention from step 16 onwards. An improvement in performance can be expected if there is a balance of local and long distance communication at every step, which will reduce contention in step 16 onwards. The Balanced Exchange (BEX) algorithm provides such a schedule. BEX is a simple modification of PEX. 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 = MOD(physical no + 1, p)
```

Table 3: Communication Schedule for BEX on 8 Processors

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

Balanced exchange consists of using the pairwise exchange algorithm with this mapping and the virtual processor numbers. The communication schedule for BEX is shown in Table 3.

The BEX algorithm has the property that in steps 0 to $p/2 - 1$, 2 processors in each cluster of size $p/2$ communicate across clusters while the rest communicate within the cluster. In steps $p/2$ to $p - 1$, 2 processors in each cluster of size $p/2$ communicate within the cluster, while the other processors communicate across clusters. In steps 0 to 15, there is no contention as in the PEX algorithm. In step $i > 15$, instead of $2^{\lfloor \lg i \rfloor}$ processors contending for $2^{\lfloor \lg i \rfloor - 1}$ links, there are $2^{\lfloor \lg i \rfloor} - 2$ processors contending for $2^{\lfloor \lg i \rfloor - 1}$ links. Hence the maximum contention for any link at step $i > 15$ is $\frac{2^{\lfloor \lg i \rfloor} - 2}{2^{\lfloor \lg i \rfloor - 1}}$ on an average. The total time taken by BEX is

$$T_{BEX} = \sum_{i=1}^{15} [\alpha + L\,\beta_{ex}] + \sum_{i=16}^{p-1} [\alpha + L\,max(\beta_{ex}, \frac{2^{\lfloor \lg i \rfloor} - 2}{2^{\lfloor \lg i \rfloor - 1}} \beta_{sat})]$$

which can be simplified to

$$T_{BEX} = (p-1)\alpha + L\,[15\beta_{ex} + (p-16)\,max(\beta_{ex}, \frac{2^{\lfloor \lg i \rfloor} - 2}{2^{\lfloor \lg i \rfloor - 1}} \beta_{sat})]$$

4.5 Performance of the Complete Exchange Algorithms

Figure 10 compares the communication time of the four complete exchange algorithms on a 32 node CM-5. The message size is varied between 0 and 2048 bytes. The LEX algorithm performs much worse than the other algorithms because of the single destination bottleneck, so we do not consider it any further. For small message sizes, the performance of PEX, REX and BEX is virtually indistinguishable. However, for large message sizes, PEX performs

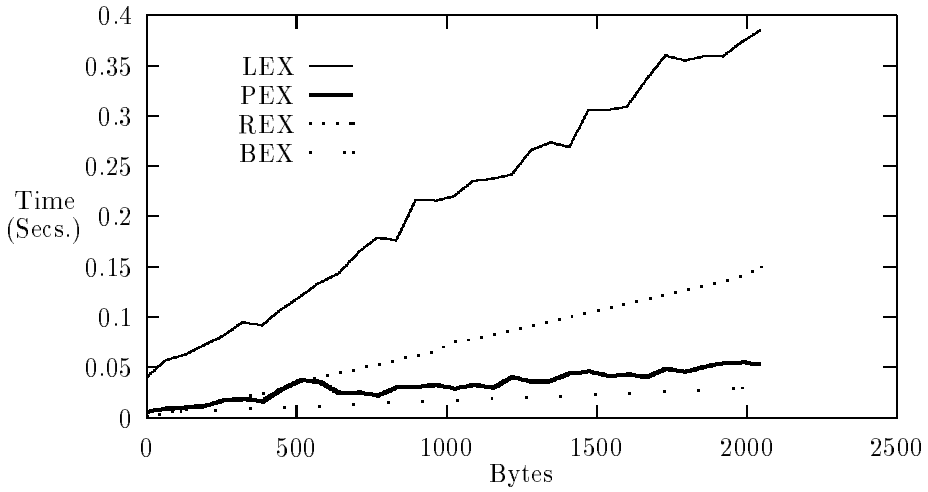

Figure 10: Complete Exchange Algorithms on 32 nodes

much better than REX and BEX performs better than PEX. This is because of the following reasons. First, even though the number of steps in REX is only $\lg p$, as compared to $p-1$ steps in PEX, the message size in REX remains constant at $L \times p/2$, whereas the size of each message in PEX is L. Also, REX uses a store-and-forward approach in which a message is sent from source to destination processor through one or more intermediate processors. Sending a message from source to destination through k intermediate processors costs k times more than sending it directly. In addition, each node needs to buffer and reshuffle data 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 measured the communication times for several machine sizes. Figures 11 and 12 show the communication times for a machine size up to 256 processors for algorithms REX, PEX and BEX. Clearly for messages of size 0 byte, REX performs better than PEX and BEX because there is no data shuffling involved and it has only $\lg p$ exchanges compared to $p-1$ exchanges in PEX and BEX. For

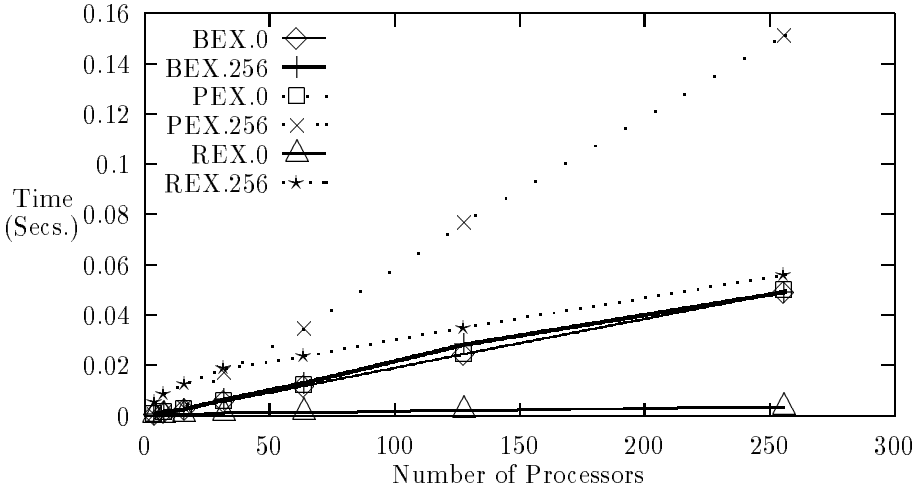

Figure 11: Complete Exchange on different number of processors (message sizes = 0, 256 bytes)

messages of size 256 bytes, PEX performs better than REX for small number of processors 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 a message size of 512 bytes and small number of processors, BEX and PEX perform better than REX. But for large number of processors, REX performs the best.

5 Scheduling Irregular Communication Patterns

An irregular problem is one in which the pattern of data access is input-dependent [10, 5]. 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 at compile time. 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

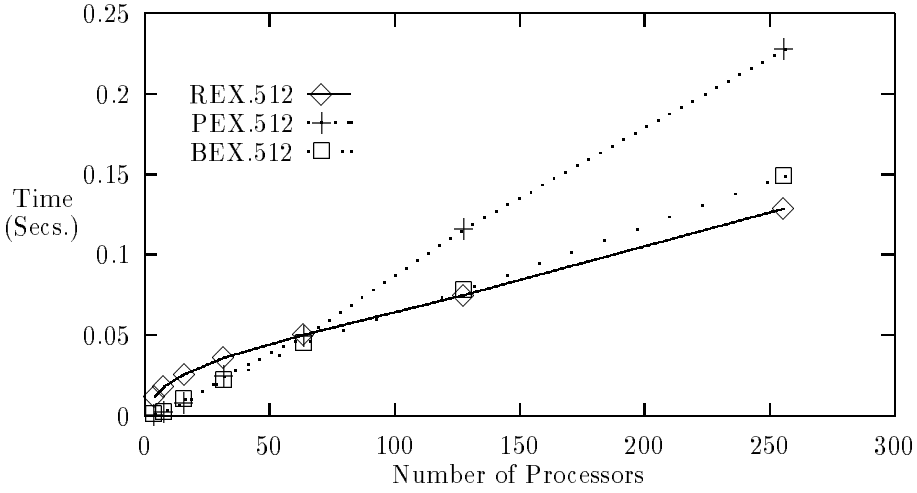

Figure 12: Complete Exchange on different no. of procs. (message size = 512 Bytes)

dynamics. To optimize communication between processors, the communication patterns in these problems can be detected and scheduled at runtime. In this section, we discuss algorithms for scheduling irregular communication patterns on the CM-5. Similar work on a hypercube architecture, using the *crystal_router*, is described in [5]. The performance effects of irregular communication patterns on the CM-2 have been studied in [12].

We 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 synthetic communication patterns as well as those arising in real problems such as the Conjugate Gradient Solver and the Euler Solver for different 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. LS, PS and BS are modifications of the LEX, PEX and BEX algorithms to take into account the irregular communication. 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.

In the greedy scheduling algorithm, each processor first uses a greedy strategy to determine the processors it has to communicate with at every step. The greedy strategy consists of selecting the next available processor in order of processor numbers, with which it needs to communicate. This schedule is then used 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.

5.1 Performance Comparison

In many problems, 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 define the percentage of non-zero entries in the communication matrix as the *communication density*. We have created synthetic communication patterns with different communication densities of 10%, 25%, 50% and 75%, 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 4. We see that the linear scheduling algorithm performs the worst in all cases because of the single destination bottleneck. 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 5. The

Table 4: Perf. of Scheduling Algorithms for Synthetic Irregular Patterns on 32 Procs.

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	1.595	2.018	3.538	4.569	5.478	7.120	6.653	8.817
Balanced	1.704	2.194	3.287	4.242	5.263	6.834	6.561	8.775
Greedy	1.476	1.905	2.781	3.599	5.201	6.792	7.739	10.31

Table 5: Perf. of Scheduling Algorithms for Real Irregular Patterns on 32 Procs.

Algorithms	Time (ms.)				
	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

table shows the communication time for each algorithm, the average number of bytes transferred in each problem and the communication density. The communication density varies from 9% in the conjugate gradient 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 in the Euler solver for a mesh with 545 vertices to 643 bytes in 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.

6 Some Scientific Applications

In this section, we study the communication performance of some scientific applications, such as two-dimensional FFT and Gaussian Elimination, on the CM-5.

Table 6: Performance of 2D FFT (Time in sec.)

	32 Processors				256 Processors			
	Scheduling Algorithm				Scheduling Algorithm			
Array Size	Linear	Pairwise	Recursive	Balanced	Linear	Pairwise	Recursive	Balanced
256×256	0.215	0.152	0.112	0.114	4.340	0.076	0.077	0.076
512×512	0.845	0.470	0.467	0.470	4.750	0.120	0.120	0.120
1024×1024	3.135	2.007	2.480	2.005	5.968	0.314	0.313	0.312
2048×2048	14.780	9.032	9.245	8.509	18.087	1.738	2.160	1.668

6.1 Two-dimensional FFT

A two-dimensional FFT of size $N \times N$ can be implemented by first performing a one-dimensional N-point FFT of each row followed by a one-dimensional N-point FFT of each column of the intermediate result. We implemented a 2D FFT algorithm with the array distributed along rows. Each processor performs a 1D FFT of the rows in its partition (no communication), followed by a transpose of the intermediate results and finally a 1D FFT of rows (because the transpose converts columns to rows). The communication required for the transpose is a complete exchange. So we implemented it using the four complete exchange algorithms described in Section 4. The performance of this 2D FFT on various sizes of data is shown in Table 6. The results obtained with the complete exchange algorithms embedded within a two-dimensional FFT routine are consistent with those obtained for the complete exchange algorithms individually in Section 4. As we can observe, for a large size 2D FFT, almost linear (relative) speedup can be obtained when going from 32 to 256 processors for all algorithms except LEX.

6.2 Gaussian Elimination

We have used the row-column-oriented algorithm with partial pivoting which is described in [14]. Gaussian Elimination requires a reduction operation to compute pivots in each iteration and a broadcast/multicast to distribute pivot element and row. Note that the CM-5 architecture provides hardware support for reduction operations through its control network. The cost of the reduction and communication operations required for Gaussian Elimination is given in Table 7 [3]. These times can be used to estimate the performance of the Gaussian Elimination algorithm. The execution time of each iteration is multiplied by

Table 7: Cost of operations of Gaussian elimination on 32 node CM-5 (time in ms.)

Operation	64×65	128×129	256×257	512×513
Reduction double with max	0.042	0.042	0.042	0.042
Reduction int with max	0.006	0.006	0.006	0.006
Broadcast from node	0.917	1.680	3.206	6.252
Computation	0.266	1.057	4.213	16.823
Time per iteration	1.231	2.785	7.467	23.123

Table 8: Estimated and measured time (ms)

Matrix Size	64×65	128×129	256×257	512×513
Estimated Time	78.8	356.5	1911.6	11839.0
Measured Time	72.5	353.45	1848.0	10948.6

the number of iterations to obtain the estimated time. There are N iterations for matrix size of $N \times (N+1)$. The measured results are compared with the estimated results in Table 8 and are found to be quite accurate. Detailed performance results of Gaussian Elimination on the CM-5 are given in [3].

7 Conclusions

This paper presented an experimental benchmarking study of the communication capabilities of the CM-5. Specifically, we gave results for communication overhead as a function of message size, distance, number of messages and contention. We considered the scheduling of dense communication patterns like complete exchange and also of irregular communication patterns. Finally, we examined the communication aspects of parallel algorithms for two-dimensional FFT and Gaussian Elimination.

The startup time of the data network was observed to be 64 microseconds. We also observed that the communication time for messages that are a multiple of 16 bytes is lower than for messages that are not, and therefore for better performance, the user should pad messages to make them a multiple of 16 bytes. For small number of messages, high bandwidth can be sustained even in large system configurations. However, as the message size and the

number of messages increases, link contention can degrade the performance. Processors within a cluster of 16 processors, can exchange among themselves without any contention, but in the case of larger clusters, the effect of contention is noticeable.

We studied the communication overhead of four complete exchange algorithms. For large number of processors, the Recursive Exchange algorithm performs the best because it requires only $\lg p$ steps. For small number of processors and small message sizes, the performance of PEX, BEX and REX is almost indistinguishable. However, for large message sizes on small number of processors, BEX performs the best because it maintains a balance of local and long distance communication at every step. REX does not perform well in this case because the overhead of large message size and data shuffling outweighs the savings in the number of communication steps.

For irregular communication patterns, the greedy algorithm performs the best when the communication density is less than 50%. The balanced scheduling algorithm performs the best when the communication density is higher than 50%.

We implemented the two-dimensional FFT using the four complete exchange algorithms discussed in this paper and the results obtained are consistent with those expected for these algorithms. The performance of Gaussian Elimination was found to be almost the same as the estimated performance.

Acknowledgments

This work was sponsored in part by DARPA under contract no. DABT63-91-C-0028 and in part by NSF grant MIP-9110810. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. We also thank the referees for their helpful comments.

References

[1] Bokhari, S., "Complete Exchange on the iPSC/860", *ICASE Technical Report 91-4*, 1991.

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

[3] Bozkus, Z., Ranka, S., and Fox, G., "Modelling the CM-5 Multicomputer", *Proceedings of Frontiers '92*, October 1992, pp. 100–107

[4] *CM-5 Technical Summary*, Thinking Machines Corporation, November 1992.

[5] Fox, G. et al., *Solving Problems on Concurrent Processors Vol I*, Printice Hall, 1988.

[6] Furmanski, W., and Fox. G., "Hypercube Communication for Neural Network Algorithms", *Caltech Report* $C^3P - 405$, February 1987.

[7] Johnsson S. L., and Ho, C., "Matrix Transposition on Boolean n-cube Configured Ensemble Architectures", *SIAM Journal of Matrix Analysis and Applications*, July 1988, pp. 419–454.

[8] Johnsson, S. L., and Ho, C., "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Transactions on Computers*, September 1989, pp. 1249–1268.

[9] Mavriplis, D., "Three Dimensional Unstructured Multigrid Solver for the Euler Equations", *Proceedings of AIAA* 10^{th} *Computational Fluid Dynamics Conference*, June 1991.

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

[11] Ponnusamy, R., Choudhary, A., and Fox, G., "Communication Overhead on CM-5 : An Experimental Performance Evaluation", Proceedings of Frontiers of Massively Parallel Computation 92, October 1992, pp. 108–115.

[12] 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.

[13] Seidal, S., 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.

[14] Wu, M., and Shu, W., "Performance Estimation of Gaussian-Elimination on the Connection Machine", *Proceedings of International Conference on Parallel Processing*, 1989.