# Communication Overhead on CM-5: An Experimental Performance Evaluation

*Ravi Ponnusamy*
*Alok Choudhary*

**CRPC-TR92255**
**March 1992**

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS - 252

# Communication Overhead on CM5: An Experimental Performance Evaluation

by
Ponnusamy, R., Choudhary, A., and Fox, G. C.

submitted to
*Frontiers '92*

March 1992

**Syracuse Center for Computational Science**
**Syracuse University**
**111 College Place**
**Syracuse, New York 13244-4100**
**<sccs@npac.syr.edu>**
**(315) 443-1723**

*Frontiers '92*

# Communication Overhead on CM5: An Experimental Performance Evaluation*

Ravi Ponnusamy Alok Choudhary Geoffrey Fox
Northeast Parallel Architectures Center
3-212, Center for Science and Technology, Syracuse University, Syracuse, NY 13244-4100

## Abstract

*In this paper we present experimental results for communication overhead on the scalable parallel machine CM-5. We study communication latency, sustainable bandwidth for simple messages as a function of message size and multiprocessor size, impact of multiple messages on the communication cost, and effects of contention on the communication cost. We also study the performance of communication intensive operations such as complete exchange and broadcast using several algorithms.*

## 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 cost 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.

Important metrics to measure the communication capabilities of a distributed memory parallel computers include latency (the time to start a message and send it out on a link), sustainable bandwidth with and without contention, effect of locality, and communication cost to perform common global operations such as broadcast and complete exchange.

This paper presents an experimental study of the communication capabilities of the Connection Machine 5 (CM-5) that was recently introduced by Thinking Machines Corporation [1]. Similar studies have been performed for other parallel machines such as Intel iPSC/2 [7], Simult 2010 [8], Intel iPSC/860 [2, 3]. Section 2 presents some details of the CM-5 and outlines the experimental setup. Section 3 presents results for messages from size 0 bytes (to compute latency) up to 10Kbytes. Also described in Section 3 are the effect of distance on the communication cost and the impact of multiple messages and contention on the communication cost. Section 4 contains the performance of several algorithms to perform complete exchange for various message and machine sizes. Section 5 presents communication overhead for performing broadcasts using two algorithms. Finally, conclusions are presented in Section 6.

*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.

## 2 Machine Details and Experiments

### 2.1 The CM-5

The CM-5 machine is, a scalable multiprocessor system, recently introduced by the Thinking Machines Corporation [1]. It can be scaled up to 16K processors. CM-5 supports both SIMD and MIMD programming models. Each node on the CM-5 is a SPARC processor which can operate at a peak 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 communications such as global reduction operations, parallel prefix operations and processor synchronization. The data network supports point-to-point communication. Its network topology is a tree-based structure as shown in Figure 1. Both data and control networks have peak bandwidth of 20 MBytes/Sec. The maximum bandwidth is obtained when communication takes place among nodes in the same cluster of four processors. The data routing nodes are also SPARC 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. CM-5 router employs random routing scheme, and therefore, the packets may be received in random order. The

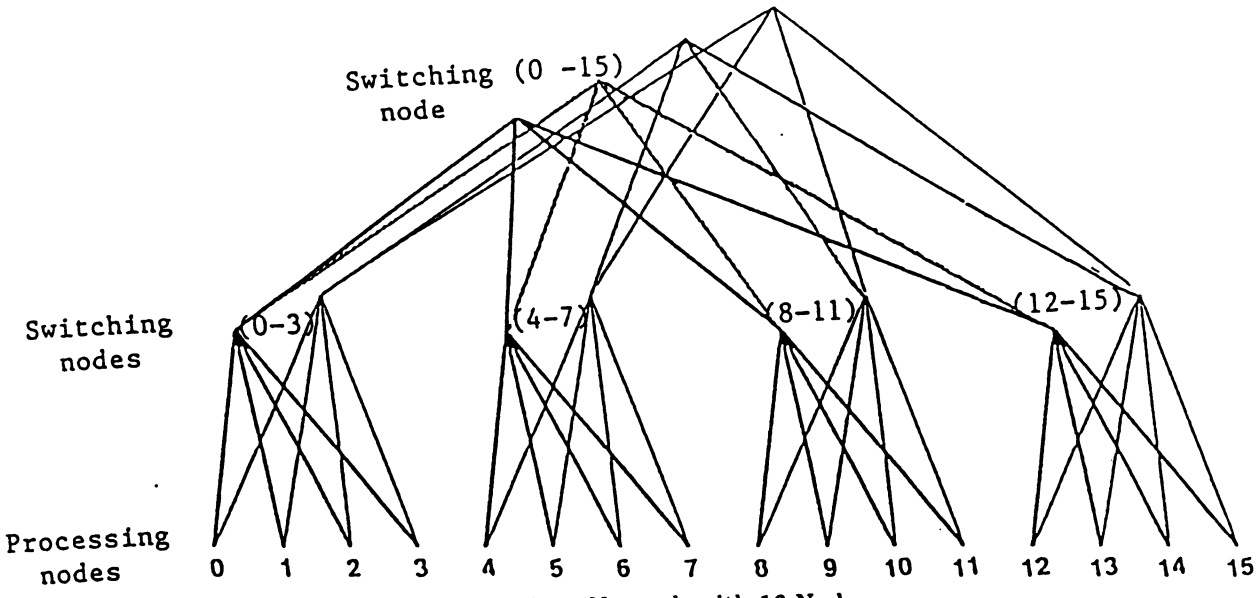

Figure 1: Data Network with 16 Nodes

data network guarantees system-wide bandwidth of 5 MBytes/sec no matter where in the system the data is being sent. Further details of CM-5 can be found in [1].

## 2.2 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 and the results presented in this paper.

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 10Kbytes. 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 affects 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 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. Since, currently only blocking send is available, for an $N$ processor system, only $N/2$ messages or exchanges can be performed simultaneously.

5. The fifth experiment studies complete exchange (which is equivalent to all-to-all personalized communication [10]) for various message sizes and multiprocessor sizes using three algorithms.

6. Finally, single source broadcast is studied using two algorithms for several message and multiprocessor sizes.

An average of 1000 repetitions of each experiment were performed to compute communication cost. The precision of the CM5 clock is 1 microsecond. For simple and multiple sends, the communication time was computed as half of the communication time for a round-trip message (the same technique has also been used in [2]).

# 3 Message Size and Path Length

## 3.1 Message Size

In the first experiment, we study the communication time for a single message of varying sizes to another node in the same cluster of processors (message set = {(0,1)}. This represents the shortest possible distance a message would travel. Figure 2 shows the communication times for messages of size 0-100 bytes. The communication time is computed by echoing the message from the destination node to the original sending node and then dividing the time by 2.

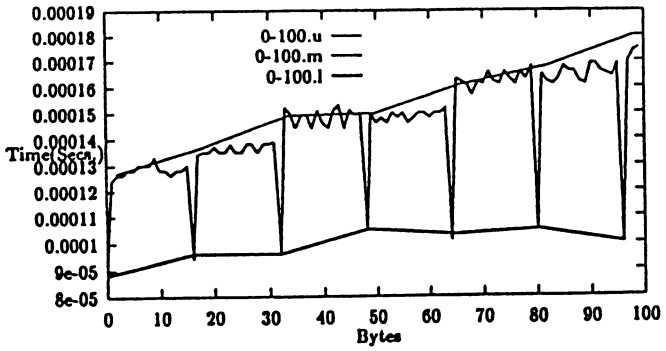


Figure 2: Message Size Varying from 0 to 100 Bytes

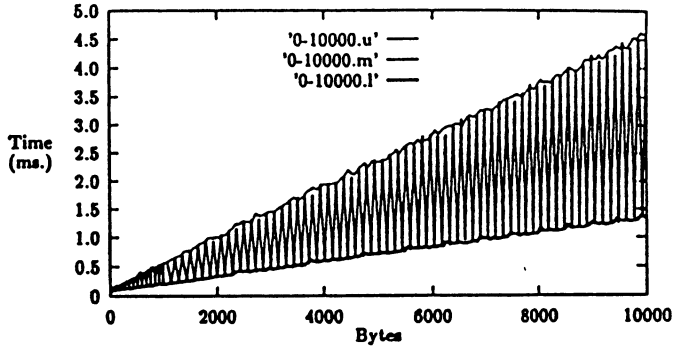


Figure 3: Message Size Varying from 0 to 10000 Bytes

A message of size 0 byte gives the communication latency, which is observed to be 88 microseconds. An interesting observation from the figure is that the communication time is different for messages that are multiple of 16 bytes than those that are not a multiple of 16 bytes. Messages that are multiple of 16 bytes follow the lower edge of the envelope, and others follow the upper edge of the envelope in Figure 2. As stated earlier, a message in 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 contains 16 bytes of user data, the overhead of processing it is smaller than the overhead of processing a message otherwise. Hence, the communication overhead incurred will be smaller if a user made the message size a multiple of 16 bytes (by padding it, if required).

The above behavior is observed consistently for messages of sizes between 100 and 10,000 bytes as shown in Figures 3. The upper edge of the envelopes in both figures represents the communication times for messages whose sizes are not integral multiple of 16 bytes, and the lower edge of the envelopes represents the communication times for messages whose sizes are integral multiple of 16 bytes. The communication time varies linearly as a function of the size of the message along both edges. The following two equations approximately describe the communication times as a function of message size for communication within a cluster of 4 processors.

$$t_{comm} = 0.126 \times l + 88 \ (in\ \mu secs.) \qquad (1)$$

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

$$t_{comm} = 0.45 \times (l - 1) + 126 \ (in\ \mu secs.). \qquad (2)$$

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

## 3.2 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 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 4 shows communication times for a message to traverse different distances in a 512-node CM-5. The x-axis values 1,2,3,4 and 5 correspond to a message being sent by processor 0 to a destination processor in a cluster of 4, 16, 64, 256 and 1024 processors. For the sake of clarity we have shown graphs for four message sizes, 16, 496, 4080 and 9968; a representative message size from each range used in the previous experiments. The plots for size 16 and 9986 bytes represent an envelope for the communication times as a function of distance for all size messages between those two limits.

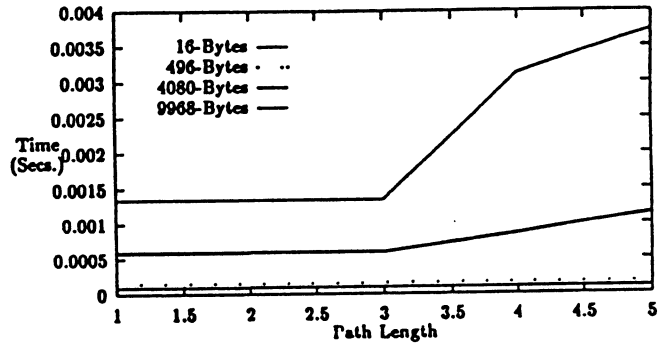


Figure 4: Impact of Path Length for Various Message Sizes

As we can observe, for small size messages, the communication time is not very sensitive to distance. However, as the message size increases, the communication time increases rapidly beyond a path length of 3. In other words, as the locality of message pass-

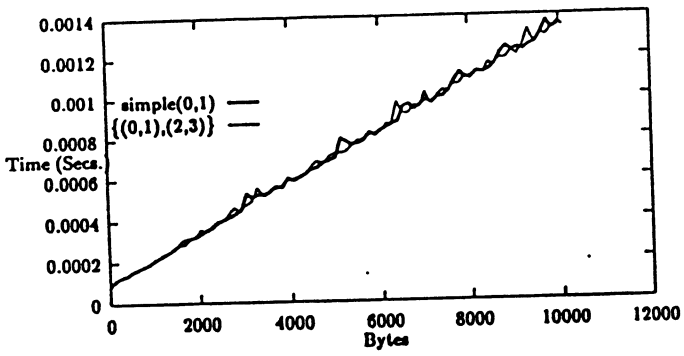


Figure 5: Comparing a simple send with double sends in a group of 4 processors

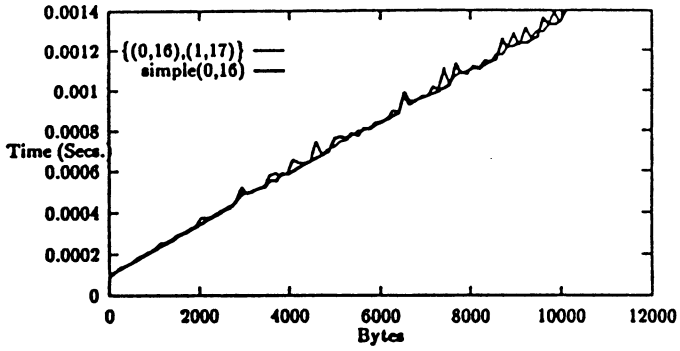


Figure 6: Comparing a simple send with a double send in a group of 64 processors

ing diminishes, the communication time starts to rise rapidly. Furthermore, as the message size increases, the rate of increases in the communication time also increases.

## 3.3 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 group of four processors. The motivation behind these experiments is to check if the communication node 0-3 (which is the parent of nodes 0 through 3) can sustain two sets of messages with the same speed. Note that CM-5 provides two paths for communication within a cluster, and it employs randomized routing. Therefore, it is expected that this experiment give the same performance as the simple send experiments in the previous subsection.

Figure 5 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). Again, we observe that there is no significant difference between the performance of SS and DS.

## 3.4 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 each message must go through the common parent node of a cluster to reach the corresponding destination processor in the neighbor-

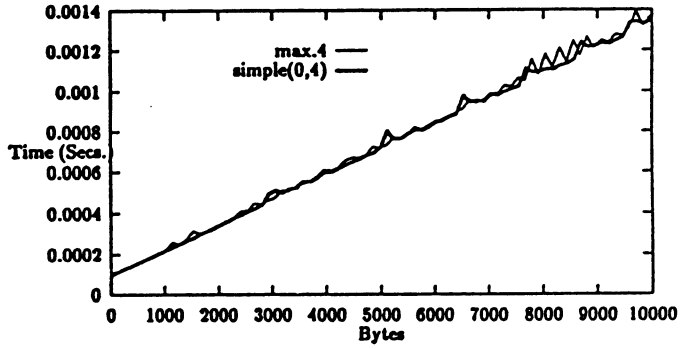


Figure 7: Comparing a simple send with maximum number of messages in a group of 8 processors

ing cluster, there is a possibility of contention. Note that since currently only synchronous communication (blocking send) is supported, for each message sent, a corresponding node should have posted a receive. Therefore, at any time, in a partition of $N$ processors, maximum possible number of simultaneous messages is $N/2$.

In the first experiment, shown in Figure 7, each processor in the first cluster of processor 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 sizes up to 10Kbytes. Hence, the presence of possible contention in small clusters does not adversely affect the communication performance in small clusters.

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

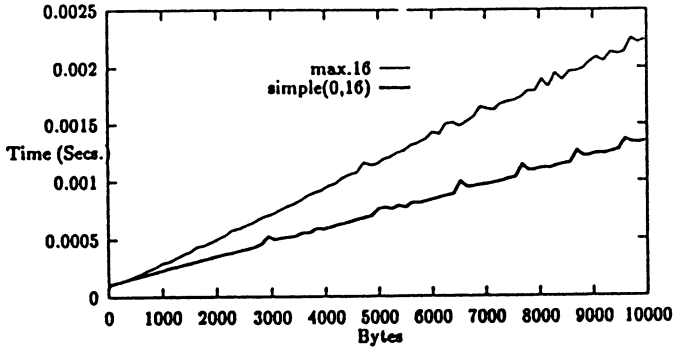


Figure 8: Comparing a simple send with maximum number of messages in a 32 processor System

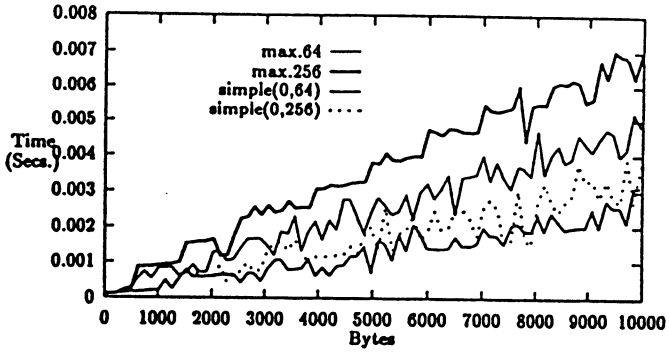


Figure 9: Comparing a simple send with maximum number of messages in 256 and 512 processor Systems

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 communication times for the above message set with that of a simple send from the source cluster to the destination cluster. 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 an increase in the message size.

Figure 9 shows the communication cost for two experiments. In the first one, all processors in a group of 64 processors communicate with the corresponding processors in another group of 64 processors (i.e., machine partition size is 128 processors and the communication set is {(0,64), (1,65), ..., (63,127)}); and in the second experiment, 256 processors in one group communicate with the corresponding processors in another group of 256 processors (machine partition size 512 nodes and communication set is {(0,256), (1,257), ..., (255,511)}). The contention effects can be clearly noticed. For a partition of size 256 processors, the contention results in degradation in performance by 66% compared to a simple send, and for a partition of size 512 processors, the corresponding degradation is 85%.

# 4 Complete Exchange

Complete exchange is a common operation encountered in computations such as matrix transpose, ADI integration and two-dimensional FFTs [3, 9]. In the next set of experiments, we implemented three different algorithms for complete exchange on the CM-5; namely, Linear Exchange (LEX), Pair-wise Exchange (PEX) and Recursive Exchange (REX).

## 4.1 Linear Exchange

The LEX algorithm is shown in figure 4.1. This is the simplest of the three 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 other processor except itself. Note that in the current version of CM-5, only synchronous communication is supported. 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 $i+1$. In such a case, performance improvements in the LEX algorithm may be expected.

```
for j= 0, nproc - 1
  if (mynumber == j) then
    for k= 0, nproc
      if(mynumber ! = k) then
        receive(k)
      end if
    end for
  else
    send(j)
  end if
end for
```

Figure 10: Linear Exchange Algorithm

## 4.2 Pairwise Exchange

The pairwise algorithm is shown in Figure 4.2. 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 an 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. PEX algorithm is better than the LEX algorithm in terms of utilizing the bandwidth of the network and reducing the processor idle

```
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 11: Pairwise Exchange Algorithm

time. The algorithm has been used in other studies such as in [6, 3].

### 4.3 Recursive Exchange Algorithm

The recursive exchange algorithm (REX) is a $log_2 N$ step algorithm for a size $N$ multiprocessor. Each message is of size $n \times N/2$ for an exchange involving $n$ bytes per processor. The algorithm is shown in Figure 4.3. Although this algorithm takes less number of steps than the other two algorithms, the amount of data transmitted in each step is much higher. Furthermore, since it is a store-and-forward algorithm, each step incurs additional overhead of reshuffling data [9].

### 4.4 Performance Comparison

Figure 13 compares the communication time of the three exchange algorithms for a 32 node CM-5 partition. The message size was varied between 0 and 2048 bytes. As expected, the LEX algorithm performs much worse than the other two algorithms. Hence, we did not consider the LEX for further analysis. For small message sizes, the performance of PEX and REX is virtually indistinguishable on this scale. However, for large message sizes, PEX performs much better than REX. This is because of the following two reasons. First, even though the number of steps in REX is only $log_2 N$, 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 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 .

In the next experiment, we selected a few message sizes in different ranges, and collected the communication times for several machine sizes. Figures 14 and 15 depict the communication times for CM-5 of size up to 256 processors for algorithms REX and PEX. Figure 14 shows times for messages of size 0 bytes and 256 bytes, and Figure 15 shows times for messages of size 512 and 1920 bytes.

```
bytes = Size/2
for i = 1, log N
  k = N/pow(2, j)
  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)
  endif
end for
```

Figure 12: Recursive Exchange Algorithm

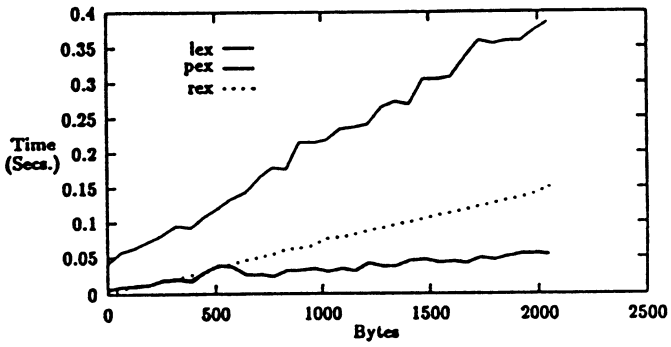


Figure 13: Complete Exchange Algorithms on 32 nodes

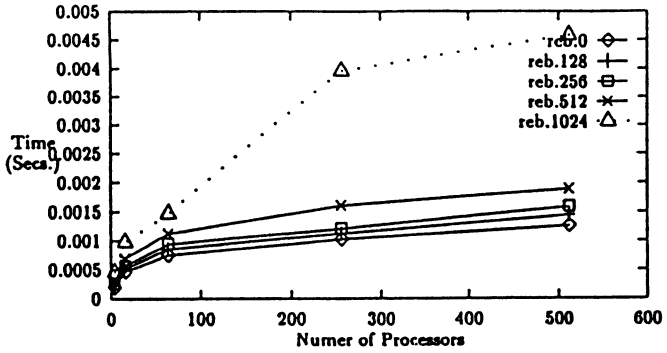


Figure 18: Recursive Broadcast Algorithm for Varying Sizes of Nodes

rithm as a function of multiprocessor size for various message sizes. For small messages, the broadcast time does not increase rapidly as the multiprocessor size increases. However, as the message size increases, the broadcast time increases more rapidly. In fact, beyond a certain size, it is better to break up a message into two (or more) smaller messages before broadcasting.

## 6 Conclusions

This paper presented experimental results for communication overhead on the CM-5. It is observed that the communication latency of the data network is 88 microseconds. We also observed that the communication cost for messages that are multiple of 16 bytes is much smaller than messages that are not, and therefore, for better performance, a user should pad messages to make them a multiple of 16 bytes. For small number of messages, high bandwidth can be sustained in large multiprocessors. However, as the message size increases and the number of messages increases, the contention can degrade the performance by as much as 90%. Hence, for large messages, locality is very important.

We also studied the communication overhead of three complete exchange algorithms. For small message sizes, the Recursive Exchange algorithm performs the best, especially for large multiprocessors. However, for large message sizes, the Pairwise exchange algorithm is preferable.

Finally, we studied two algorithms for one-to-all broadcast; namely, the Linear broadcast algorithm and Recursive broadcast algorithm. Linear broadcast does not perform well at all. The recursive broadcast algorithm performs well. The most interesting and important conclusion is that for large message sizes, it is better to split a broadcast message into two or more smaller messages depending on multiprocessor and message size.

## References

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

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] Chittor S. and Enbody R., Performance analysis of Symult 2010's interprocessor communication network. TR CPD-89-19, Michigan State University, CS, 1989.

[9] 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.

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

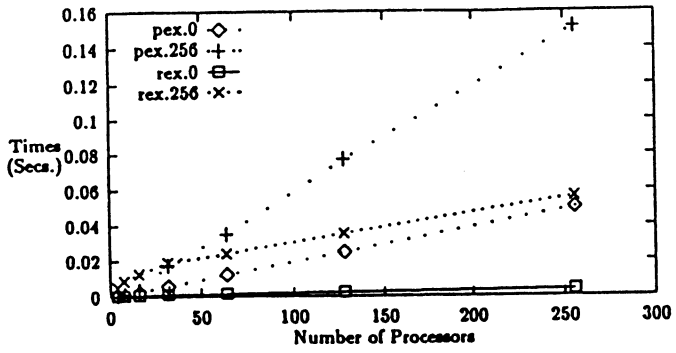


Figure 14: Complete Exchange Algorithms for Varying Multiprocessor Sizes

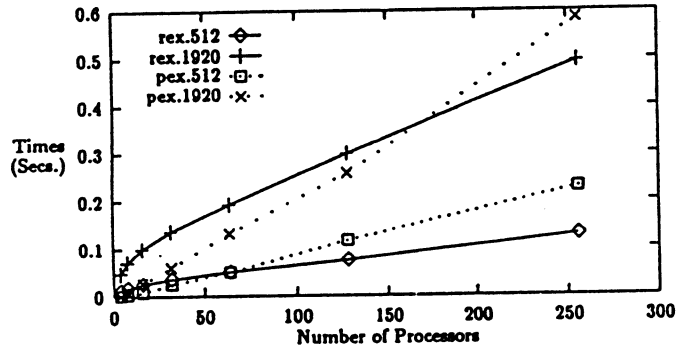


Figure 15: Complete Exchange Algorithms for Varying Multiprocessor Sizes

Obviously, for messages of size 0 byte, REX performs better than PEX for all multiprocessor sizes because there is no data shuffling involved. For messages of size 256 bytes, PEX performs better than REX for small multiprocessor sizes because the overhead of message size and number of steps dominate for REX. As the partition size increases, overhead of the larger number of messages dominates the overhead of larger messages and reshuffling in REX, and therefore, REX performs better. Figure 15 shows the communication overheads when message size is increased further. The crossover is again seen for message size 512, but the crossover point has moved to the right as compared to that for message size 256 (Figure 14). This is because, beyond a certain size, the overhead of shuffling and larger message size is significantly larger than the savings provided by a much smaller number of communication steps.

```
for j = 1, logN
  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 16: Recursive Broadcast Algorithm

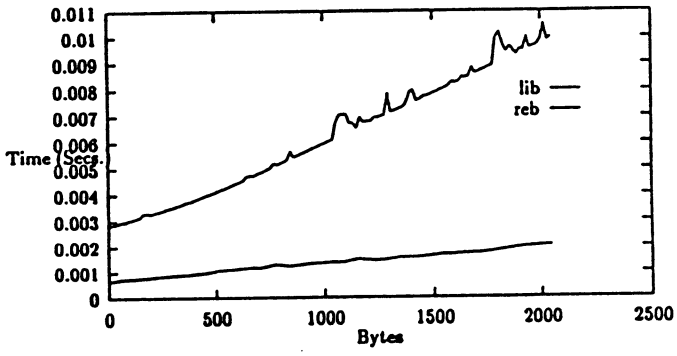


Figure 17: Broadcast Algorithms on 32 nodes

## 5 Broadcast

Broadcast is a very common communication primitive encountered in many applications. We consider one-to-all broadcast (also known as single source broadcast) [10]. This section presents performance of two broadcast algorithms; namely, Linear Broadcast (LIB) and Recursive Broadcast (REB).

The LIB is the simplest 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, there are $log_2 N$ 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. Figure 5 presents the algorithm.

Figure 17 presents the performance of the two algorithms 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.

Figure 18 presents performance of the REB algo-