Modeling the CM-5 multicomputer [1]

Zeki Bozkus, Sanjay Ranka and Geoffrey Fox

School of Computer Science

4-116, Center for Science and Technology

Syracuse University

Syracuse, NY 13244-4100

zbozkus@npac.syr.edu

ranka@top.cis.syr.edu

gcf@npac.syr.edu

[1]This work was supported in part by NSF under CCR-9110812 and DARPA under contract # DABT63-91-C-0028. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

Abstract

This paper describes the performance evaluation and modeling of the CM-5 multiprocessor. We provide a number of benchmarks for its communication and computation performance. We have also benchmarked many operations, like scans and global reduction, that can be performed using special hardware available on the CM-5. Most of these operations are are modeled using regression on appropriate parameters.

These models are used to predict the performance of Gaussian Elimination on the CM-5. Comparative timing on the Intel Touchstone Delta machine are also provided.

We also describe how to efficiently embed a mesh and a hypercube on a CM-5 architecture and provide timings for some mesh and hypercube communication primitives on the CM-5.

1 Introduction

The CM-5 is a distributed memory multiprocessor. The processors are interconnected using three networks: data network, control network and the diagnostics network. The data network is useful for processor to processor communication for bulk transfer. The control network, in general, is useful to perform operations which require the participation of all the nodes simultaneously, such as broadcasting and synchronization. Thus communication between two nodes can be performed by using the data network or the control network. The diagnostic network is for fault diagnosis/maintenance of the system. This paper is restricted to the control network and the data network.

This paper presents results of a set of benchmark programs run on the CM-5 machine[1]. Our study of the current CM-5 performance has two main goals. The first goal is to benchmark the CM-5 for its performance. The second goal is to model the CM-5 primitives in a fashion that it will be useful for estimating the performance of applications on CM-5. These primitives are modeled by linear performing regression using SAS, a statistical package widely available. The parameters for each primitive are chosen appropriately.

The rest of the paper is organized as follows. Section 2 briefly describes CM-5 system.Section 3 gives computation benchmarks.Section 4 discuss the CM message-passing library. Section 5 and 6 gives communication benchmarks from node to node and host to node respectively. Section 7 addresses some of the global operations provided by the CM-5. Section 8 gives brief summary of communication performance. Section 9 and 10 discusses about simulating mesh and hypercube on quad tree. Section 11 presents performance estimation of CM-5 and compare with the Intel Touchstone Delta. Finally, section 12 gives the conclusion of this paper.

[1] *Note to the referees: the current version of the paper is limited to the scalar version of the CM-5. It is expected that within the next few months the CM-5 would be upgraded to the floating point vector units increasing the computational performance significantly. We expect to add performance results of vector units at that time.*

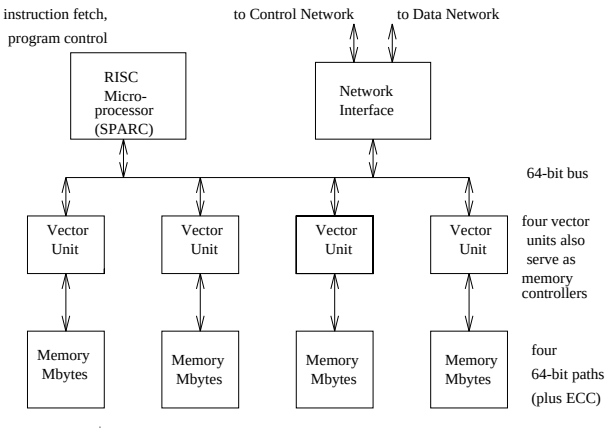

Figure 1: CM-5 processing node with vector units.

2 System Overview

This section gives a brief overview of the CM-5 system. Our overview is divided into two main sections: the processing unit and the communication networks.

2.1 Processors

Each node is a Sparc microprocessor with four vector units of a CM-5 system and 32 Mbytes of memory (Figure 1.) The Sparc processor has a clock of 33Mhz. A 64Kbyte cache is used for both instruction and data. It has separate integer and floating point registers. The Sparc processor is rated at a peak performance of 22 Mips or 5 MFlops. It performs the instruction fetching, controls the processing engines for vector units (in the full version of CM-5) and manages the communication with other system components via the network interface. The vector units perform one memory operation and one arithmetic operation per cycle at 16Mhz (half the rate of the Sparc). It can perform a multiply and add in one cycle. Thus the peak performance of each vector unit is 32 Mflops for multiply-add and 16Mflops for add or multiply.

Thus a processing node with four vector units can give a peak performance of 128 MFlops (for multiply-adds) and 64 Mflops for multiplies or adds alone.

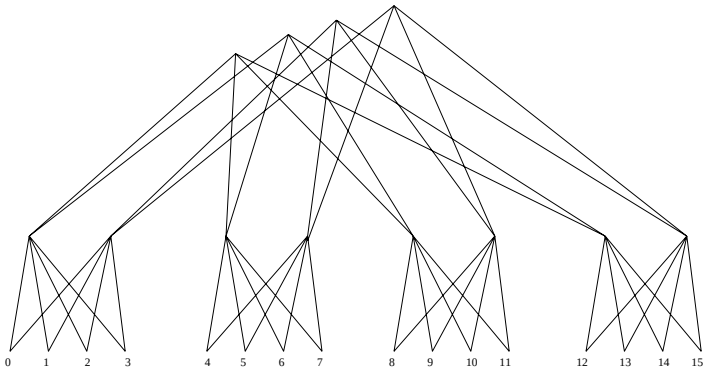

Figure 2: Data network with 16 nodes.

2.2 Networks

Every node in the CM-5 is connected to two interprocessor communications networks, the data network and the control network. There is a separate network for carrying out diagnostics.

2.2.1 Control Network

The control network provides hardware support for common cooperative operations. These operation include integer sum, max, min, and, or, xor, parallel prefix and segmented parallel prefix. Further, it provides barrier synchronization. The control network has a relatively low latency. This combined with a high bandwidth provides fast execution of the above operations.

2.2.2 Data Network

The data network is a message passing point to point routing network. The network topology is a quad tree based structure with a network interface at all the leaf nodes. The CM-5 is designed to provide a peak bandwidth of 5 MBytes/s for point to point communication between any two nodes in the system. However, if the destination is within the same group of 4 or 16, it can give a peak bandwidth of 20 Mbytes/s and 10 Mbytes/s respectively[1].

Figure 2 and 3 show the network for 16 nodes and 64 nodes respectively. The CM-5

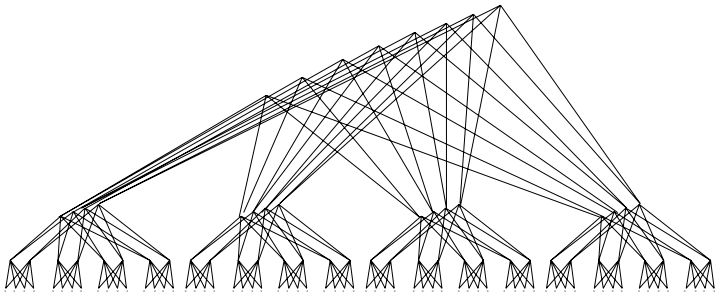

Figure 3: Data network with 64 nodes.

network is redundant in nature and is fault tolerant. The routing is deadlock free and the conflict arbitration is fair [2].

2.3 Test System

Our experiments were performed on 32 node CM-5 at Northeast Parallel Architecture Center at Syracuse and a 512 node CM-5 at The Army High Performance Research Center at University of Minnesota.

All the measurements reported in this paper were done using programs written in C. The timings were estimated by measuring the time for m repetitions of the experiment and dividing the time by m. The value of m was varied from 1 to 100 as required for the accuracy of measurement.

3 Computation Benchmarks

We implemented a series of tests to measure the arithmetic speed of the Sparc processor for integer and floating point operation. A comparison with the timings on the Intel Delta processor i860 is also provided at table 1. The current Sparc chip has performance 2 to 3 times slower than i860 processor (running at 40 MHz) .

Operation	CM-5 (μs)	DELTA (μs)
short +	1.22	0.54
long +	1.35	0.62
float +	1.57	0.68
double +	1.88	0.69
short *	1.75	0.69
long *	1.90	0.62
float *	2.19	0.62
double *	2.05	0.68

Table 1: Time for arithmetic operation on CM-5 and Delta

4 CM-5 Communication Library CMMD

The CM message-passing library, CMMD, provides facilities for cooperative message passing between processing nodes. CMMD supports a programming model which is referred to as Host/Node programming. This model involves program running on the on the host, while independent copies of the node program run on each processing node. CM-5 can be reconfigured into many partitions. The host acts like a partition manager (PM). It begins execution by performing needed initializations and then invoking the node programs.

5 Basic Node to Node Communication

This section describes the results for basic node to node communication. Message passing routines support two types of messages: standard messages in which bytes are in normal sequential order and vector messages in which elements (many bytes) are stored with strides (a fixed amount of space). Most of the routines support the same primitives for both types of messages. We only performed experiments for standard messages. The performance of these primitives for vector messages should be similar.

5.1 Send and Receive for Regular messages

We wanted to measure the time necessary for sending a message from one node to another. We measured this time for node 0 to all the other nodes in the system. Since, one way messages are difficult to measure, this time is estimated by dividing the round trip time by 2. The round trip time is calculated using the following algorithm. Node 0 starts the clock and sends a message to node i ($i \neq 0$). Node i waits for a message form node 0 and echoes this message back. Node 0 stops the clock after receiving the message. The send and the receive used on the CM-5 are both blocking in nature.

5.1.1 Neighbour communication

The best performance communication is between two neighbours in all networks. We measured the message passing time between node 0 to node 1 on the CM-5. If the buffer is improperly (word) aligned the current implementation is relatively slower (by a factor of 4 !) for moving data from node memory into Network interface. Figure 4 shows sending different size of messages from Node 0 to Node 1. But figure 5 shows sending word aligned messages (divisible by 16) from Node 0 to Node 1. The time can be modeled by the following equation

$$T(l) = 73.39 + 0.126 * l \qquad microseconds \tag{1}$$

For comparison, the corresponding equation for the Delta at [3] is

$$T(l) = 72 + 0.08 * l \qquad microseconds$$

Thus the startup time is approximately 73.39 microseconds and transfer rate is proportional to 0.126.

Figure 6 shows that the transfer rate for the aligned buffer is around 8 Mbytes/sec. This bandwidth is significantly lower than the peak bandwidth of 20 Mbytes/sec [2]. In the current CM-5 implementation, system communication bandwidth isn't limited by the ability of the node processor to shove data into network. Assembler codes can achieve something like 18 Mbytes/sec moving data from one node's registers to another's. But C with CMMD library programs tend to go slower, partly because the C compiler's output is never as efficient

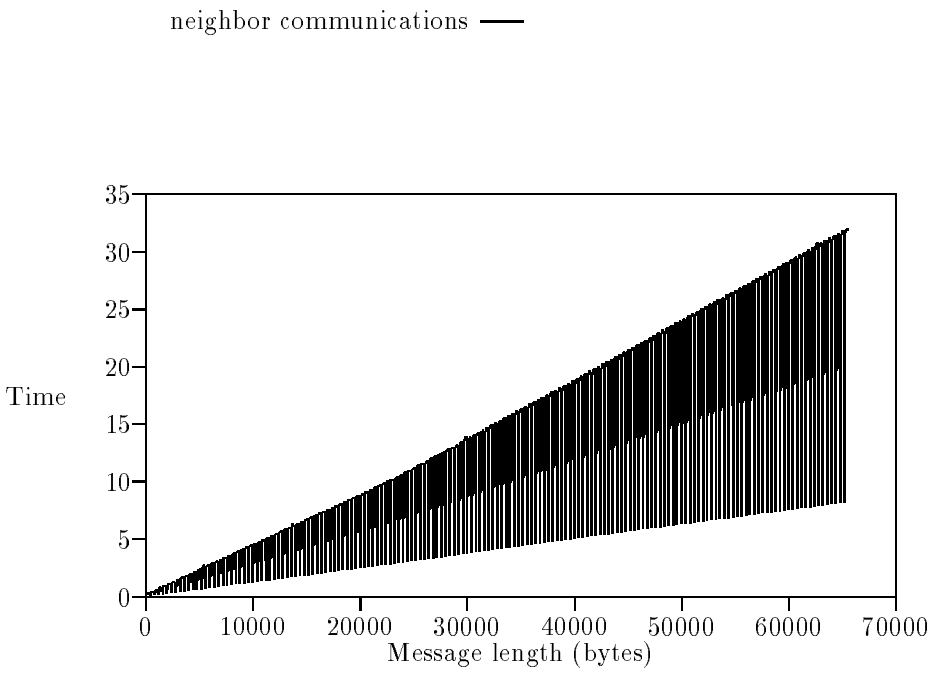

Figure 4: Neighbour communication between node 0 to node 1 with unaligned message length (not have to be divisible by 16)(time is milliseconds).

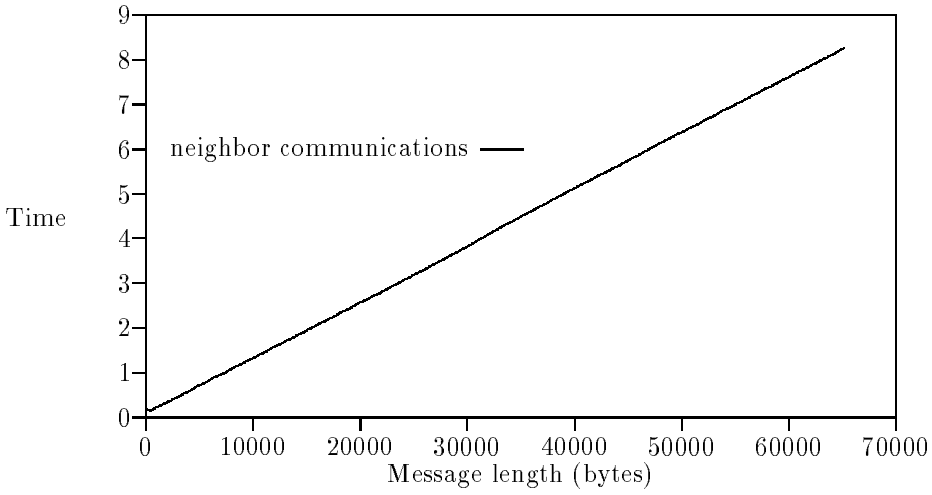

Figure 5: Neighbour communication between node 0 to node 1 with word aligned message length (divisible by 16) (time is milliseconds).

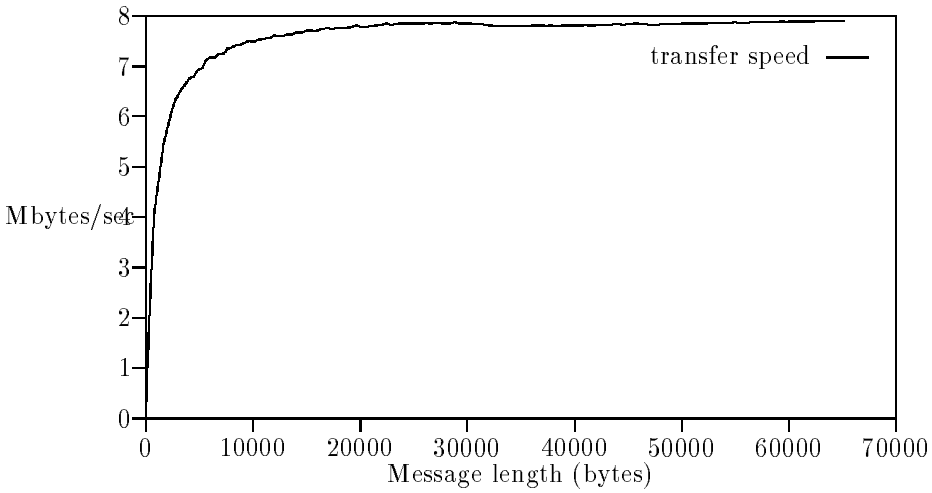

Figure 6: Transfer rate for neighbour communication between node 0 to node 1 with word aligned messages

as hand-crafted assembler, partly because it's tricky to convince the compiler to generate double-word data transfer instructions[4]. It is expected that future versions of the library will improve this raw communication bandwidth.

5.1.2 Non-neighbor communication

We measured the communication time from node 0 to every other nodes using the same strategy as the previous section. The results for a message of size 16 and 2000 bytes are presented in figure 7 and 8. It shows that the difference in time to communicate between neighbour and non-neighbour is negligible. Again, this could be attributed to the fact that the bandwidth is limited by how fast the Sparc processor (using the C program) can pump data into the network interface.

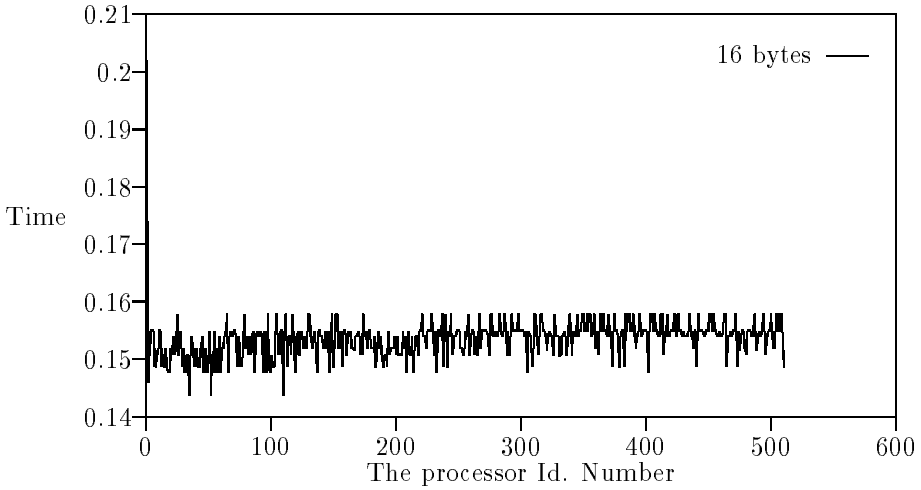

Figure 7: Communication from node 0 to every other node on a 512 nodes CM-5 for a message of 16 bytes (time is milliseconds).

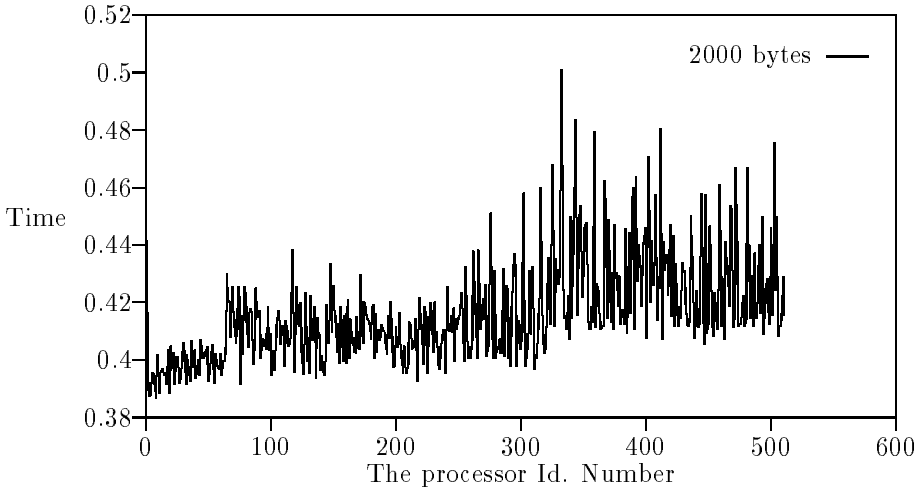

Figure 8: Communication from node 0 to every other node on a 512 nodes CM-5 for a message of 2000 bytes (time is milliseconds).

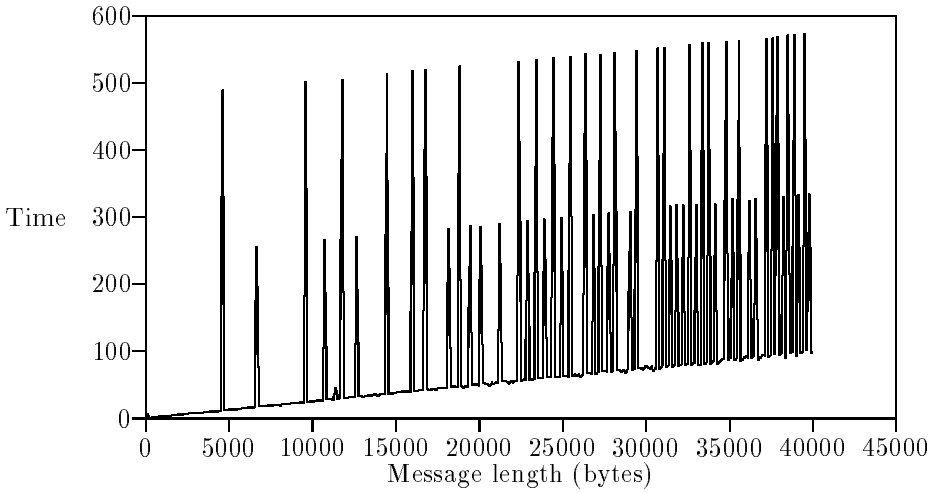

Figure 9: Communication time from host to node 0 (time is milliseconds).

6 Host to Node communication

The CMMD library also provides a rich set of global operations. These include broadcast, parallel prefix, and synchronization operations provided by the underlying hardware, All nodes must cooperate in these operations and call them before the execution of the primitive is completed. This provides implicit synchronization. These primitives use the control network for performing their operations.

6.1 Host to node Send and Receive

Figure 9 shows the time to send a message from host to a node. Figure 10 shows the transfer rate from host to node. Host to node transfer rate is around 0.4 Mbytes/sec. This is quite small as compared to the node to node transfer rate of 8 Mbytes/sec.

Partition Manager's (PM) VME bus become a bottleneck for movement of data from the PM's memory into the data router and into nodes. This is due to the slower speed of the bus and the non exclusive use of the VME bus (with Unix).

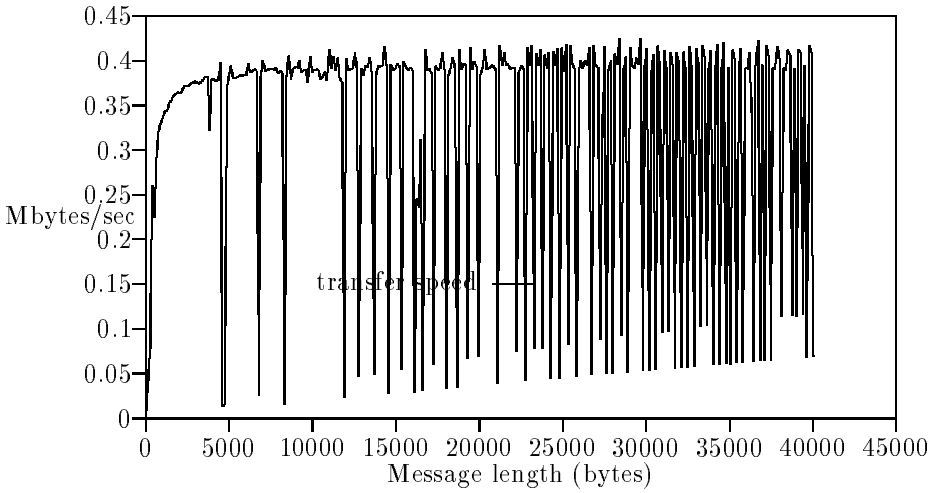

Figure 10: The transfer rate from host to node 0

Node-host communications are also constrained by the fact that the node is communicating outside its partition when it's talking to the host. Communication outside a node's partition requires a trap into the node OS, which involves more overhead than the direct writes required to communicate inside a partition [4].

6.2 Broadcast from host

In a broadcast operation, messages are sent from the host to all nodes. The host broadcasts the entire contents of the buffer to all nodes. All nodes receive the same amount of data simultaneously. The host and all the nodes must take part in a broadcast. The node processors need to wait for a hardware signal that the entire broadcast is complete before performing any other operation.

Figure 11 shows broadcast timing for 32 nodes and 512 nodes. The time for broadcast for 32 and 512 nodes is very close to each other. Thus, the CM-5 has a scalable architecture for broadcast. Equation 2 and 3 gives the relationship between message length and time of broadcast operation for 32 and 512 nodes respectively.

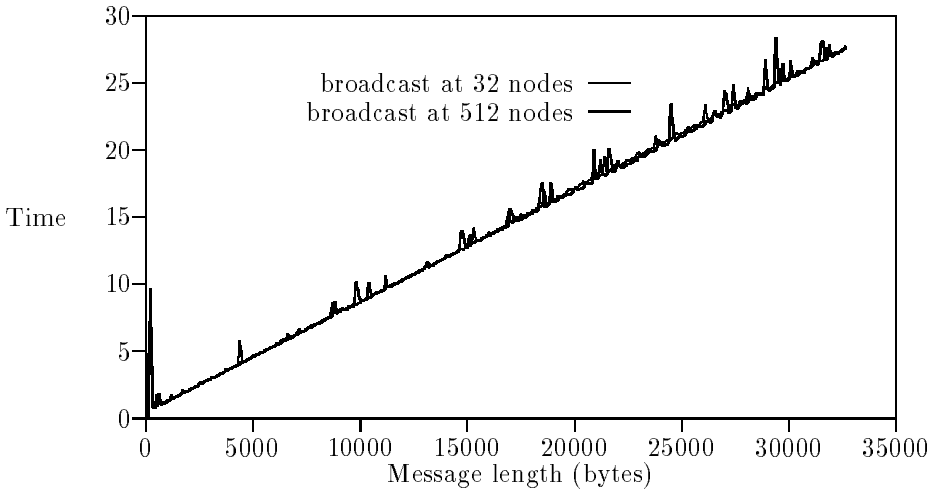

Figure 11: Broadcasting timing at 32 nodes and 512 nodes (time is milliseconds).

$$T(l) = 527 + 0.83 * l \qquad microseconds \tag{2}$$

$$T(l) = 542 + 0.83 * l \qquad microseconds \tag{3}$$

Broadcast is faster than the host-node CMMD_send because it uses the Control Network. Thus by using CM-5 broadcast library, each node can receive messages from host at a transfer rate of approximately 12 Mbytes/sec. (Figure 11).

6.3 Distribute from host to nodes

This primitive communicates an array (from the host) to all the nodes such that each nodes receives a consecutive block (of equal size). The partitioning is done similar to the block decomposition of the Fortran D Language specification [5]. This primitive is very useful when the host performs I/O and input has to be block partitioning among nodes. Figure 12 shows the performance of this primitive. We derive the Equation 4 and 5 for 32 and 512 nodes respectively.

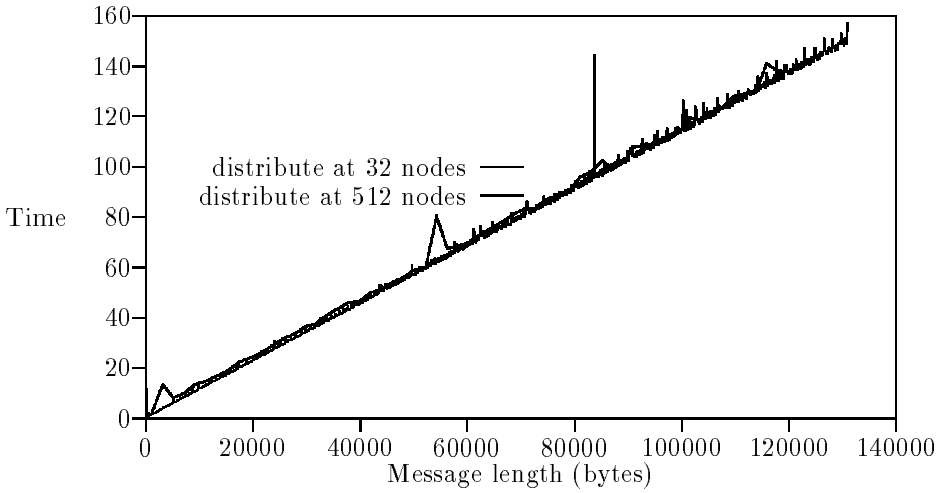

Figure 12: Distribution from host to nodes on 32 and 512 CM-5. (Time is milliseconds)

$$T(l,p) = 360 + 1.15 * l \qquad microseconds \tag{4}$$

$$T(l,p) = 2600 + 1.13 * l \qquad microseconds \tag{5}$$

7 Global Operations

Scans, reductions, and concatenation are some of the global operations provided by the CMMD library. Given a buffer containing a value in each node, these global computations operate simultaneously on the buffer set to perform such tasks as

- summing the value across all the nodes
- finding the largest or smallest value
- performing a bitwise AND, OR, or XOR

Operation	type	add	max	min	ior	xor	and
scan	int	10.14	10.14	9.51	10.48	10.47	9.81
scan	unit	10.13	9.61	9.81	10.50	10.82	10.16
scan	double	47.57	54.16	55.91	*	*	*
segmented scan	int	10.80	10.79	10.25	11.12	10.98	10.46
segmented scan	uint	11.13	10.50	10.48	11.48	11.48	9.29
segmented scan	double	123.26	125.25	132.87	*	*	*
reduce with host	int	74.50	74.52	73.56	73.92	74.51	74.58
reduce with host	uint	78.39	79.21	77.76	79.65	78.46	77.96
reduce with host	double	187.51	194.88	197.52	*	*	*
reduce only nodes	int	6.63	6.95	6.29	6.95	6.96	6.63
reduce only nodes	uint	6.98	6.31	6.32	7.30	7.31	6.80
reduce only nodes	double	39.48	42.21	46.76	*	*	*

Table 2: scan, segmented scan and reduction with host (result on the host and nodes) reduce only host (result only on the nodes) operation on 32 nodes. Time unit is microseconds. * : represents an undefined operation.

7.1 Scan Operation

A scan (a parallel prefix operation) creates a running tally of results in each processor in the order of the processor identifier. Assuming that the $A[j]$ represents the element A in the jth processor and $R[j]$ represents the result R in the jth processor. Then a scan (with an add) performs the following operation

$$R[i] = \sum_{j=0}^{i-1} A[j] \quad 0 \leq i < P$$

where P is the number of processors.There are several versions of scan with behaver similar to above.

Table 2 shows the performance of scan operation for a 32 node CM-5.

7.2 Segmented Scans

In a segmented scan, independent scans are computed simultaneously on different subgroups (or segments) of the nodes. For a detailed description of segmented scans, the reader is referred to [6]. The segments are determined at run time by an argument called the *sbit* which is a segment bit. $sbit = 1$, represents the beginning of a new segment. Table 2 shows the performance of a segmented scan operation on a 32 node CM-5 assuming *sbit* is 1 with a probability of 0.1.

7.3 Reduction Operation

A reduction operation takes as input a value in every processor and outputs a single value, either in every processor or in the host processor. The operation performed can be one of the following : *add*, *max*, *min*, *ior*, *xor* and *and*.

Table 2 gives the time required for the reduction operation assuming that the result is on the host and nodes or the result is on the nodes. The host in the reduction operation slows down the operation by an order of magnitude.

7.4 Concatenation Operation

Concatenation appends the value from each processor to the values of all preceding processors (in processor identifier order). CMMD provides two versions of concatenation: one concatenates across the nodes only, and writes the resulting value into a buffer on every node. The other concatenates values from every node into a buffer on the host. The result of the concatenation is always ordered from the lowest to highest node [7].

Assuming that each processor has N elements, and there are P processors. Suppose processor i contains a vector $V_i[0 \cdots N-1]$. The global concatenate operation computes a concatenation of the local list in each of the processors. The resultant vector $R[0 \cdots NP-1]$ is stored in the host or in every node.

$$R[j] = V_{j \; div \; N}[j \; mod \; N]$$

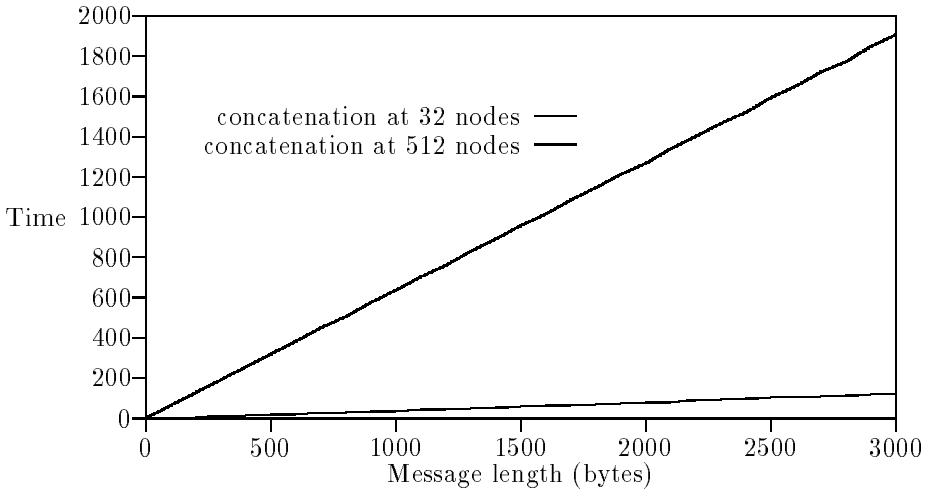

Figure 13: Concatenation with result at the 32 nodes and 512 nodes (time is milliseconds).

Figure 13 and 14 shows the time required for concatenation using 32 and 512 nodes with the result at the nodes and result at the host respectively. The length of the resultant vector is proportional to the product of N and P. From figure 13, it is clear that the time for concatenation on 512 nodes is 15.36 ($\approx$ 16) times larger than 32 nodes. From figure 13 we approximately derive the equation 6 for concatenation operation at nodes.

$$T(l,p) = p + 1.2 * p * l \qquad microseconds \tag{6}$$

l is message length and p is the number of nodes in that partition. Clearly, concatenation operation is scalable.

$$T(l,512)/T(l,32) \approx 16$$

However when the result is to be stored on the host the timings are irregular and larger than the case when results are stored on the nodes. This can be mainly attributed to the reasons described in section 6.

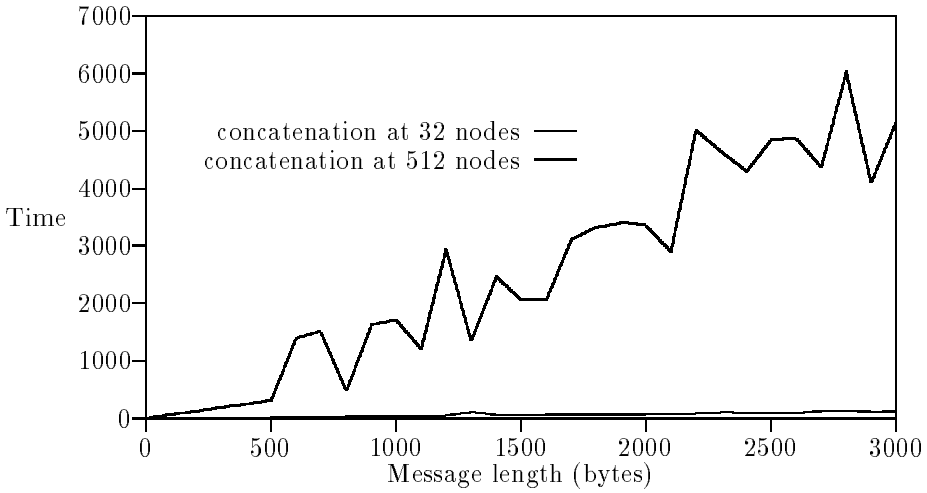

Figure 14: The concatenation with result host and nodes at 32 nodes and 512 nodes (time is milliseconds).

7.4.1 Broadcast from one node to all nodes

When we write SPMD style programming, one of the common kind of communication is to broadcast from one node to the rest of node. Figure 15 shows performance of this primitive at CM-5. We derive the following Equation 7 and 8 for 32 and 512 nodes CM-5.

$$T(l) = 19.38 + 1.49 * l \qquad microseconds \tag{7}$$

$$T(l) = 5.0 + 1.46 * l \qquad microseconds \tag{8}$$

8 Summary of Performance Models for Collective Communications

Table 3 presents a summary of the performance modeling of all the global communication primitives for CM-5. This table shows that CM-5 is scalable for these operations.

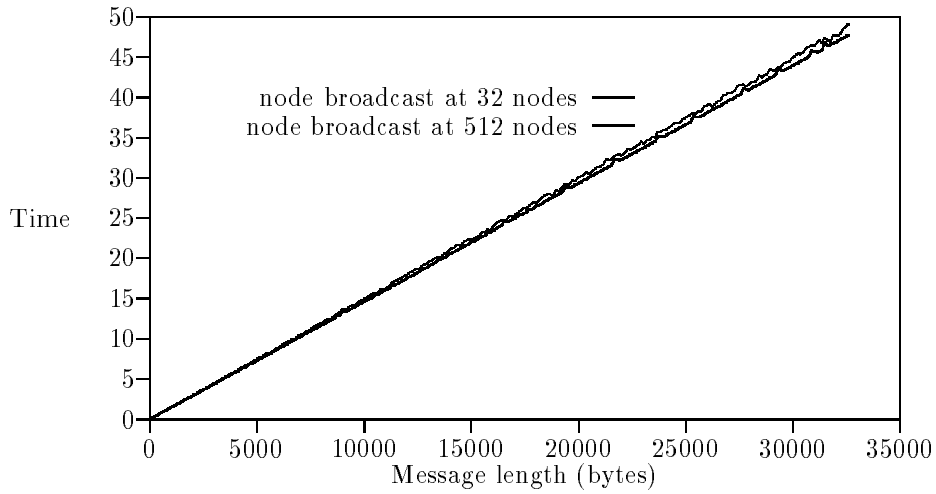

Figure 15: Broadcasting from one node to all the other nodes on a 32 and 512 nodes CM-5 (time is milliseconds).

Communication Type	Derived Formula	
Neighbour	T(l) = 73.39 + 0.126*l	microseconds
Broadcast from Host (32 nodes)	T(l) = 527 + 0.83*l	microseconds
Broadcast from Host (512 nodes)	T(l) = 542 + 0.83*l	microseconds
Broadcast from node (32 nodes)	T(l) = 19.38 + 1.49*l	microseconds
Broadcast from node (512 nodes)	T(l) = 5.0 + 1.46*l	microseconds
Distribute from node (32 nodes)	T(l) = 360 + 1.15*l	microseconds
Distribute from node (512 nodes)	T(l) = 2600 + 1.13*l	microseconds
Concatenation to Host	T(l,p) = p + 1.2*l	microseconds

Table 3: Summary of all derived equations for communication on CM-5

9 Mesh simulation

A wrap-around mesh (torus) can be embedded into the CM-5 quad tree based architecture by using the shuffle row major mapping. This mapping preserves the locality of 2×2 and 4×4 sub meshes. Figure 16 gives two procedure for calculating the mapping of a node on a $m \times n$ mesh on a quad tree architecture and its inverse. The *from_coordinate_to_procnum* and *from_procnum_to_coordinate* procedures help every node to find out their north, east, west and south neighbour. The shuffle and unshuffle in figure 16 are bitwise operations. If $i = abcd$ and $j = efgh$ then $shuffle(i, j)$ returns $aebfcgh$. $unshuffle$ is a reverse operation of $shuffle$. The size of $unshuffle$ input is $2log_2(m)$ bits long. Tables 4 shows the timings of 16×32, 8×64, 4×128, 2×256 mesh simulation on a 512 node CM-5 .

```
from_coordinate_to_procnum(i,j)
{
    jprime = j mod m;
    block = j / m;
    procnum = m*m*block + shuffle(i, jprime);
}

from_procnum_to_coordinate(procnum, i, j)
{
    block = procnum / (m*m);
    procnumprime = procnum - block * m*m;
    (i, jprime) = unshuffle(procnumprime);
    j = jprime + m*block;
}
```

Figure 16: An algorithm to simulate mesh on CM-5

mesh	message	NORTH		EAST		WEST		SOUTH	
size	size	max	min	max	min	max	min	max	min
16x32	16 K	15.21	15.00	16.80	15.57	21.84	20.82	15.15	15.00
16x32	32 K	29.72	29.55	32.15	19.42	32.66	19.73	19.65	19.39
16x32	64 K	56.43	48.36	76.45	57.30	66.55	63.61	56.70	49.49
8x64	16 K	15.01	14.91	21.72	20.81	21.18	4.26	4.42	4.11
8x64	32 K	24.50	24.26	31.83	19.15	32.57	19.17	19.18	18.93
8x64	64 K	37.64	37.04	65.10	60.60	87.32	66.86	49.28	26.26
4x128	16 K	4.22	4.15	15.88	15.34	20.60	19.95	9.62	9.95
4x128	32 K	13.36	1.70	20.62	18.99	29.23	29.00	13.75	12.93
4x128	64 K	32.60	31.57	67.62	49.12	55.74	53.76	32.89	31.64
2x256	16 K	4.07	3.05	20.43	20.34	14.74	14.64	9.40	3.05
2x256	32 K	12.72	11.46	35.74	35.48	24.61	24.39	18.57	11.45
2x256	64 K	24.89	22.47	49.05	48.66	61.95	54.75	31.55	22.90

Table 4: The timing of mesh 16×32, 8×64, 4×128 and 2×256 simulation with 512 nodes CM-5 (time is millisecond).

10 Hypercube simulation

For many computations, the communication pattern required is similar to the connections of a hypercube architecture. These included bitonic sorting, Fast Fourier Transform and many divide and conquer strategies [8]. This section discusses the time requirements for such communication pattern.

A p-dimensional hypercube network connects 2^p processing elements (PEs). Each PE has a unique index in the range $[0, 2^p - 1]$. Let $i_{p-1}i_{p-2}\ldots i_0$ be the binary representation of the PE index i. Let $\overline{i}_k$ be the complement of bit i_k. A hypercube network directly connects pairs of processors whose indices differ in exactly one bit; i.e., processor $i_{p-1}i_{p-2}\ldots i_0$ is connected to processors $i_{p-1}\ldots\overline{i}_k\ldots i_0, 0 \leq k \leq p-1$. We use the notation $i^{(b)}$ to represent the number that differs from i in exactly bit b.

We consider communication patterns in which data may be transmitted from one processor to another if it is logically connected along one dimension. At a given time, data is transferred from PE i to PE $i^{(b)}$ and from PE i to PE $i^{(b)}$.

Node i of a logical hypercube is mapped on node i of the CM-5. The communications patterns performed for a logical hypercube on the CM-5 using this mapping are shown in figure 17. The time required for swapping data along different dimensions is approximately the same for all dimensions and scales linearly with the size of the message (for messages as as large as 256K bytes). The rate of transfer is between 4.4 Mbytes/sec and 5.3 Mbytes/sec. This is close to the peak bandwidth for long range communication on the CM-5. However for messages with locality (dimension 1 and 2) the bandwidth is nowhere close to the peak of 20 Mbytes/sec. This is due to the reasons described in section 5.

11 Performance Estimation for Gaussian elimination

Modeling of the primitives is useful in estimating the performance of a given program [9]. We use Gaussian elimination to illustrate how to estimate the performance of a program by using results mentioned in the previous sections. Figure 18 is the computational intensive segment of a Gaussian elimination program. The row-column-oriented algorithm with partial

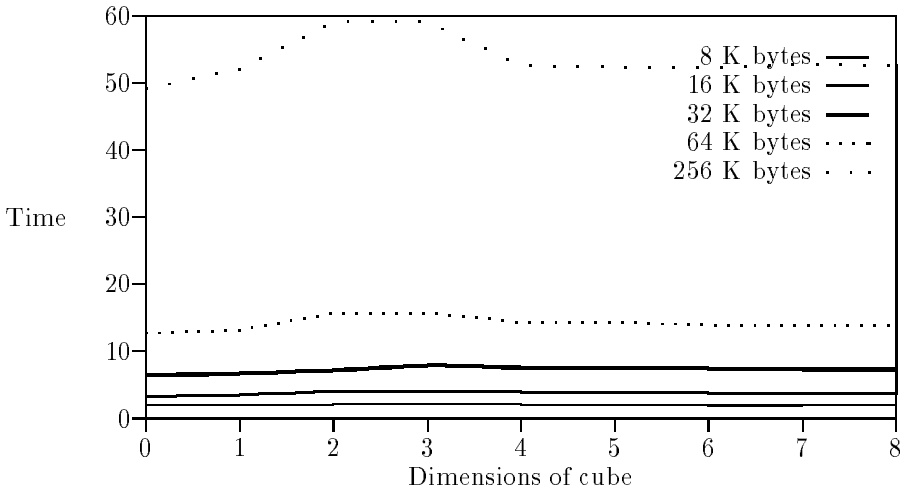

Figure 17: the timing of each level of fat tree leaves at level 0 root level 8 (time is milliseconds).

pivoting is used here [8] .

Table 5 is a portion of the relevant data derived from our benchmarking and modeling results presented in the previous sections. The execution time of each iteration is multiplied by the number of iterations to obtain the estimated time. There are N iteration for matrix size of $N \times (N+1)$. This code was executed on a 32 node CM-5. The measured results are compared to the estimated results in Table 6. Such modeling can be useful in performance prediction different algorithms. One can use this information in choosing algorithms and optimizing program codes. One can automate performance estimation at compile time by using cost function of each primitives instead of real operations.

We now compare CM-5 to the Intel Touchstone Delta Mesh for Gaussian elimination. The Delta system is a message-passing multicomputer, consisting of an ensemble of individual and autonomous nodes that communicate across a two-dimensional mesh interconnection network. It has 513 computational i860 nodes, each with 16 Mbytes of memory and each node has peak speed of 60 double-precision Mflops, 80 single-precision Mflops at 40 MHz.

```
double a[N/nproc][N+1]
for(i=0;i<N;i++) {
  find pivoting row
     by using two reduction operations;
  broadcast pivoting row;
  some vector computation;
}
```

Figure 18: Gaussian Elimination Code for Program Estimation.

Operation	Reference	64×65	128×129	256×257	512×513
Reduction double with max	Table 2	0.042	0.042	0.042	0.042
Reduction int with max	Table 2	0.006	0.006	0.006	0.006
Broadcast from node	Equation 7	0.917	1.680	3.206	6.252
Computation	Table 1	0.266	1.057	4.213	16.823
Time per iteration		1.057	2.787	7.469	23.130

Table 5: Cost of operations of Gaussian elimination on 32 nodes CM-5 (time is milliseconds).

Matrix Size	64×65	128×129	256×257	512×513
Estimated Time	78.9	356.7	1912.1	11843.0
Measured Time	72.5	353.45	1848.0	10948.6

Table 6: Estimated and measured time (time is milliseconds).

Operation	64×65	128×129	256×257	512×513
gopf	0.86	0.86	0.86	0.86
Broadcast from node	0.43	0.72	1.36	2.50
Computation	0.12	0.48	2.18	11.10
Time per iteration	1.41	2.06	4.40	14.46

Table 7: Individual operation of Gaussian elimination on 32 nodes The Delta Mesh (time is milliseconds).

We implemented the Gaussian elimination algorithm in both system by C + MP (Message Passing).

Table 7 show timing of individual operation of Gaussian elimination in the Delta Mesh. Comparing the results of Table 5 and Table 7, The Delta *gopf* [2] operation is slower than two reduction operations of CM-5. But the Delta is faster for broadcasting and computation.

Table 8 compares the performance on the two multicomputers. Thus, for Gaussian elimination it can be seen that the 32 nodes CM-5 without vector nodes is comparable to the 32 nodes Delta system. This is because the CM-5 (without vector units) has poor performance as compared on computation to Intel Delta Machine. However, it has a faster mechanism for performing reduction. Since large matrixes have more computation as seen on the tables, CM-5 is better for smaller matrixes and worse for large matrixes.

12 Conclusions

In this paper, we studied the performance of the CM-5 multiprocessor. We provided a number of benchmarks for its communication and computation performance. The computation performance is currently limited because the vector units are not available. In the current version the CM-5 Sparc chip has performance 2 to 3 times slower than i860. It is expected that the performance should improve by at least an order of magnitude with the

[2] The Delta function which lets user define its own function. We designed our function to find pivot row and its processor number in the Delta Mesh

Matrix Size	CM-5 32 Nodes	DELTA 32 Nodes
64×65	72.5	126.0
128×129	353.4	331.0
256×257	1848.0	1261.0
512×513	10948.6	6062.0

Table 8: Comparison of CM-5 and DELTA (time is milliseconds).

full configuration (with the vector units).

The maximum node to node transfer rate was found to be 8 Mbytes/sec for unidirectional transfer between two neighbour nodes. This was significantly lower than the peak bandwidth of 20 Mbytes/sec. It is expected that this rate is going to increase with later versions of the CMMD software. The host to node communication is significantly slower than the node to node communications. The maximum transfer rate for long messages was of the order of 0.4 Mbytes/sec. Thus algorithms which require active participation of the host and require large number of messages to be communicated from the host to nodes will, in general, have poor performance and poor scalability. We studied some collective communication patterns required for solving many interesting scientific problems (such as the mesh and hypercube). The bandwidth for hypercube type of communications was between 4 and 5 Mbytes per second. This was also true for cases when all the communication passed through the root (pseudo) of the CM-5 interconnection network. For the mesh type of communication the bandwidth was relatively lower. For most patterns the transfer rate was between 1.0 and 2.5 Mbytes/sec. The reasons of hypercube type patterns having higher transfer rate can be attributed to the following: 1) hypercube patterns require a swap between two processors; 2) hypercube patterns embed more regularly on the CM-5 interconnection network.

There are several global operations which use the control network for communication. Broadcast from the host to nodes achieves a transfer rate of 12 Mbytes/sec. Concatenation from nodes when the host is not participating requires time linearly proportional to the size of the resultant array. Concatenation requires significantly more time when the result is to

be stored at the host. The other operations, like max, min, add, xor, and etc., which use the global control network are extremely fast when the result is to be stored on the nodes and can be completed in 6 to 7 microseconds for integers and 39 to 47 microseconds for double precisions. The time are an order of magnitude higher when the result need to be stored on the host. Scans and segmented scans are useful primitives for solving many applications [6]. These operations are quite fast and can be completed in 10 microseconds for integers on 32 nodes. For double precision the times are significantly larger (a factor of 4 for scans and a factor of 13 for segmented scans).

For most primitives, the participation of the host generally degraded the total time for completion. Thus the CM-5 is more suitable for programs which require minimum participation of the host (such as the Single Program Multiply Data style of programming).

The CM-5 data and control network were found to be highly scalable. The performance figures remained constant for most operations when we evaluated similar primitives from 32 to 512 nodes. Once the vector units are added to the current version of CM-5, the computation performance along with the scalable communication provided by the data and control network and the global operations (like scans, synchronization, max etc) provided by the control network should make the CM-5 a versatile machine for solving many computationally intensive, interesting scientific and engineering problems.

A comparative study was performed between Intel Touchstone Delta and CM-5 (without vector units). We found that a 32 node CM-5 had a comparable performance to the Touchstone Delta. Although, the Sparc processor is slower as compared the i860 chip. Reduction operation is faster on CM-5 but its broadcast is slower than on the Delta

Acknowledgements

We would like to thank Steve Swartz of Thinking Machine for many clarifications regarding the CM-5. We would like to thank Professor Vipim Kumar for providing access to the CM-5 at the Army High Performance Computing Center at University of Minnesota.We would also like to thank Min-You Wu for his suggestions on improving the manuscript.

References

[1] Brond Larson. personal communications. 1992.

[2] Thinking Machines Corporation. In *The Connection Machine CM-5 Technical Summary*, October 1991.

[3] T. H. Dunigan. Communication Performance of the Intel Touchstone Delta Mesh. Technical report, Oak Ridge National Laboratory, 1992.

[4] Steve Swartz. personal communications. 1992.

[5] Geoffry Fox, Seema Hiranadani, Ken Kenndy, and etc. Fortran D Language Specification. Technical report, Rice and Syracuse University, 1992.

[6] Guy E. Blelloch. In *Vector Models for Data-Parallel Computing*. MIT, May 1990.

[7] Thinking Machines Corporation. In *CMMD Reference Manual*, Januar 1992.

[8] G. C. Fox, M.A. Johnson, S. W. Otto G.A.Lyzenga, J.K. Salmon, and D. W. Walker. In *Solving Problems on Concurent Processors*, volume 1-2. Prentice Hall, May 1988.

[9] Min-You Wu and Wei Shu. Performance Estimation of Gaussian-Elimination on the Connection Machine . *International Conference on Parallel Processing*, 1989.