Performance Evaluation of Mesh-Connected Wormhole-Routed Networks for Interprocessor Communication in Multicomputers

Suresh Chittor Richard Enbody

Department of Computer Science
Michigan State University
East Lansing, Michigan 48824-1027

ABSTRACT

Wormhole routing is becoming a popular switching technique in the design of interprocessor communication networks. In this paper, we describe a set of experiments conducted on wormhole-routed Symult 2010 systems to measure the performance of their interprocessor communication networks. We then analyze the results to predict the performance of similar systems. We first show that long path lengths in this mesh have a negligible effect on communication time. Next, we show that the contention for network channels can scale up with the system size, and that the effect is observable in current machines. We also show that resources within a node itself can become a bottleneck. Given current architectural trends, we conclude that contention for network resources, and not path length, will become the primary concern in ensuring fast communication in future multicomputers.

1. Introduction

Some of the largest and most powerful computers such as the NCUBE, the iPSC, and the Symult 2010 [16] belong to a class of parallel systems called multicomputers. They are distributed-memory, MIMD machines with nodes connected by a point-to-point interprocessor communication network (ICN). The performance of the ICN is critical to the success of multicomputers. Rapid advances have been made over the last few years in the technology of such networks, and this paper examines one important implementation.

Although the Symult computer is no longer in production, there are many lessons to be learned from this machine. The important characteristics of an implementation of a point-to-point network are the switching technique, the underlying topology of the network and the routing scheme. The switching technique determines how communication takes place between two non-adjacent nodes. A store-and-forward switching technique was used in earlier systems such as the NCUBE-1 and the iPSC-1. With store-and-forward, communication takes O (LD) time, where L is the message length and D is the distance between the communicating nodes (path length). Since, the communication time increases with D, low diameter networks were used, and parallel tasks had to be mapped properly to minimize the distance between the communicating tasks. As a result, the mapping problem was extensively studied. Second generation multicomputers attempt to address this problem using circuit switching as in iPSC/2 [11] or wormhole routing as in Symult 2010 [16], and iWarp [2] which take O ($L+D$) time [5,6]. Since L is usually much greater than D ($L \gg D$), O ($L+D$) $\approx$ O (L) which indicates that path lengths should no longer be a significant problem. Wormhole routing and circuit switching, however, suffer from their own disadvantages. In both, individual messages reserve channels and other network resources. As a result, messages must compete for resources and the resulting contention can affect the communication time significantly.

Another important characteristic of ICNs is the underlying topology of the network. Ideally, we would like to have a completely connected network, but current implementation constraints rule out such a network for large systems. Since small path lengths were very important for store-and-forward switched networks, topologies having low diameters such as hypercubes were very popular. In addition, hypercubes have several other nice features such as low degree, regularity, symmetry, and so on. The 2D mesh (or torus) is another popular topology that has similar features, but has a much higher diameter. On the other hand, meshes are much simpler, and are easier to wire. Since wormhole routing reduces the importance of path lengths, a low diameter may no longer be a critical requirement. Hence, meshes have become a viable alternative to hypercubes.

Recently, it has been shown that 2D mesh and torus networks can have wider channels and faster communication rates than hypercube networks, resulting in a higher bandwidth between nodes [5,6,12]. It has also been shown that 2D meshes and tori will have lower communication delays than hypercubes if we consider that difference in bandwidths [6]. Dally has shown that for uniformly distributed communication, 2D meshes will perform better than hypercubes even if we consider contention [6].

At present, only small, second-generation systems are available commercially, and most of the available performance results concentrate on systems with less than 100 nodes [3,8,9,13]. Recently, results of performance analysis, similar to ours, have been reported for another second generation multicomputer, the iPSC-2 [1], which uses a circuit-switched, hypercube based ICN [1]. However, the iPSC-2 can only have 128 nodes. As a result, little information is available on the performance of large 2D networks using either circuit-switching or wormhole routing.

CH2916-5/90/0000/0647/$01.00 © IEEE

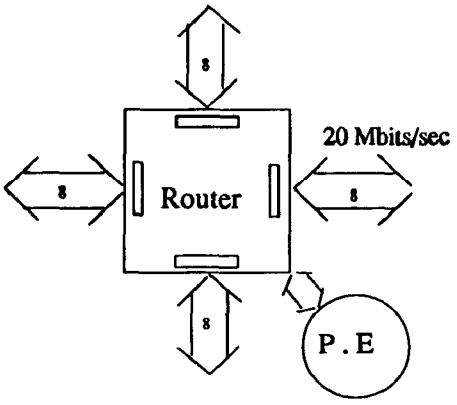


Figure 1. Symult 2010's node architecture

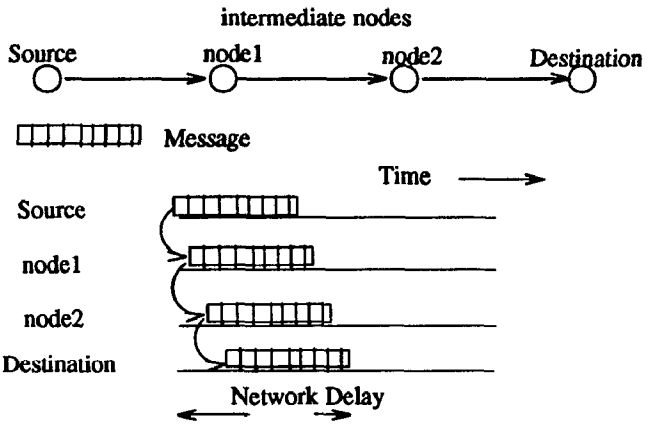


Figure 2. Pipelined flow of message flits in wormhole routing

We find that this work is significant is several ways.

(1) The experiments were run on a real system that uses a state-of-the-art, wormhole-routed network, so the results indicate the performance that can be expected at the user level.

(2) The experiments reveal the problems and bottlenecks in the current architecture. These results indicate architectural areas to be examined in future designs for improved performance.

(3) The experimental results are employed to generate performance curves which can be used to predict the increase in communication delays in future systems.

(4) We discuss the current architectural trends and how they will affect our conclusions.

The paper is organized as follows. In Section 2, we briefly describe the Symult 2010's architecture and how wormhole routing works. In Section 3, we describe the experiments and present the results. In Section 4, we describe how to generalize the results by eliminating the machine-dependent parameters. In Section 5, we consider future systems. Finally, we summarize our results and suggest future research areas.

2. Architecture

An understanding of both the architecture of the ICN of the Symult 2010 machine and wormhole routing is essential for understanding our results. Figure 1 shows the overall architecture of a node. It has a Processing Element (P.E.) which includes the processor, memory and interface to the router. The Router, called the MRC, supports interprocessor communication [7]. The MRC is similar to the Network Design Frame (NDF) [4,17] which is also designed for 2D wormhole-routed networks. The MRC has five bi-directional channels, one to each of its four neighbors and one to the local processing element. Each channel is 8 bits wide and operates at 20 Mbits/sec, yielding an overall bandwidth of 20 Mbytes/sec. The channels support bi-directional traffic by time multiplexing the same eight data lines between the traffic in opposite directions. Each channel has a parity line which will detect occasional single errors during transmission of a single flit between two adjacent nodes. A *flit* is a word of 8 bits which is the unit for communication between adjacent nodes. Unlike the iPSC-2, there is no end-to-end acknowledgment at the hardware level. The Symult 2010 router uses a fixed routing scheme that avoids deadlocks: message flits travel along row channels, and then along column channels to reach the destination node.

The flow of the message flits in wormhole routing is shown in Figure 2. Wormhole routing is similar to the virtual cut-through routing proposed by Kleinrock et al. in 1979 [10]. An intermediate node begins forwarding the data as soon as the header is received, provided that the outgoing channel is free. A message consists of a set of data flits preceded by one or two header flits. The router examines the header flit as soon as it is received to determine whether the message is for the local processing element. If not, the router will quickly determine an outgoing channel. If that channel is free, the router will forward the header and the remaining data flits as they arrive until a tail flit is received. In this way, the data flits will follow the header flit in a pipelined fashion from the source to the destination node. If the header flit encounters a busy channel, it will get blocked. All the following flits will become blocked at their current node causing the partially established path to be blocked. Only the tail flit can release the channels reserved by the header flit. All the routing functions can be implemented in hardware, and the P.E. is not involved in processing messages passing through a node. The router is simple and uses little chip area. Each channel is associated with a one-flit buffer (or at most a few flits in a queue). Large numbers of buffers and complex buffer management are not required.

We refer to a router-to-router channel as a *network channel* or *rr-channels* and a P.E to router channel as a *pr-channel*. We now proceed to describe the experiments.

3. Experiments

Experiments were run on Symult 2010 systems at Caltech under the Cosmic Environment / Reactive Kernel (CE/RK) -- a Caltech message passing environment available on several multicomputers [15]. We ran the experiments using 16 nodes (connected as a 4×4 2D mesh), 64 nodes (8×8) or 144 nodes (12×12).

We introduce this section with a list of general comments, notations and precautions that were necessary to ensure the validity of the measurements. In our experiments, we

needed to isolate the desired network performance from other system characteristics.

1. Once the programs were spawned onto the nodes, the host waited for 5 secs for spawning to complete before sending (broadcasting) the input parameters of the experiment. This ensured that intensive traffic due to program spawning will end before the start of the experiments. If we had not waited, this extraneous traffic would have interfered with the messages generated by the experiments and affected the loop times.

2. In each experiment our aim was to find the average time taken by a node to complete a given program segment. Such a segment, called the *base loop*, consists of both local computation and communication with other nodes. This loop requires 0.5 ms to 15 ms, but the clock resolution is only 1ms so the base loop was repeated N times. Traffic due to the broadcast of the experiment parameters interfered with the communication in the initial loops. Hence, the first n loops were ignored. The clock reading at the beginning of the $n+1$th loop was subtracted from the clock reading at the end of the Nth loop, yielding the time for $N-n$ loops, say T ms. The average loop time was then computed as $T/(N-n)$. In our experiments we found that N = 11 000 and n = 1000 are sufficient. Thus we measured the total time in msecs for 10 000 loops on each node.

3. Each node computed the average loop time, as described above, by using its own clock for each reading. This local timing avoided any problems due to the variance in different clock readings. In addition, each node executed the same program. In most experiments, the mean of the loop times determined by individual nodes was used as the final loop time to avoid idiosyncrasies of individual node hardware (or the kernel).

4. The objective of the experiments was to measure the performance of the communication network and not the Cosmic Environment functions that support messages. Hence, messages were created before the beginning of the first loop, and the same set of messages was used throughout the experiment. Messages were neither created nor destroyed once the first loop started. This strategy prevented any processing time by functions, such as xalloc or xfree (which manage message buffers), from affecting the measured times.

5. "Blocking receives" were used in all experiments to capture the delays in the communication. The receiving nodes were usually ready to receive by the time a message arrived at the node. This technique prevented any program-generated delays in accepting a message from appearing as communication delays.

6. The kernels were different on different machines, and were modified during the course of the experiments. Changes due to kernel performance were avoided as follows. Many experiments were performed in pairs one after the other; the first tested the inclusion of a specific factor under study, and the other tested the exclusion of the feature. We then calculated the difference in communication times. Timings measured on different days were never compared.

We use the following notation in this paper. The architecture consists of N nodes connected as a $\sqrt{N}\times\sqrt{N}$ 2D mesh with nodes numbered 0..N-1. Node (i,j) refers to a node located in the i^{th} row and j^{th} column; numbering is 0..$\sqrt{N}$-1. In most experiments node (i,j) will communicate with only one other node while executing the base loop. We call that node the *peer* node: peer (node (i,j)) refers to the peer of node (i,j).

In all the experiments, the base loop consisted of local computation followed by communication with other nodes. In the base loop definition, compute (t) denotes local computations (no communication with other nodes) for t ms. Communications are represented by two functions *tsend* and *trecv*

tsend (*dst*,*msg*,*msgln*) : Send a message of *msgln*, which starts at the address *msg*, to the node *dst*. The function is non-blocking.

msg = *trecv* (*src*) : Receive a message from the node *src*. The message will be at the address *msg* when the call returns. This is a blocked receive so that the call will not return until the message arrives.

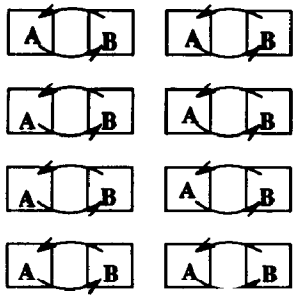
Loop_time refers to the time taken by a node for the base loop of an experiment on average. *Comp_time* denotes the time for local computations so that communication time can be determined by (*loop_time* - *comp_time*).

In each experiment, nodes are partitioned into three classes I, A and B. Nodes belonging to the the class I are idle, and do not participate in the experiment. The remaining nodes belong to either class A or B. A node and its peer usually belong to different classes.

The experiments are explained in three sections. In most experiments, nodes execute the same base loop. The positions of the communicating nodes were changed to form different communication patterns, causing changes in the communication demand on the network. In Section 3.1, we measure the performance for contention-free communication between adjacent nodes, which is the best an application programmer can achieve. The times in this section are neither affected by longer path lengths, nor by contention for channels and data paths. In the next section, Section 3.2, we concentrate on the effect of path lengths. Non-adjacent nodes communicate, but contention is still carefully avoided. In Section 3.3, several experiments are described which involve communication under contention.

3.1 Contention-free Adjacent node Communication : In the first set of experiments, communication is restricted to physically adjacent nodes. Node (i,j) will communicate with node $(i,j\hat{}1)$ where $\hat{}$ denotes the *exclusive-or* function, that is peer (node (i,j)) = node $(i,j\hat{}1)$. The N nodes of the system are therefore partitioned into $N/2$ pairs, and these pairs do not interfere with each others communication. None of the nodes are idle, i.e. none belong to the class I. Node (i,j) belongs to class A if j is even, and to class B otherwise. In the base loop, a message of L bytes is sent from an A node to its peer, and then a message of the same size is sent back from the peer [Figure 3].

Contention is carefully avoided. First notice that at any time a node will be either sending or receiving. Thus, there is traffic along only one direction on the pr_channels. The communicating node pairs do not interfere with each other. Also,



Class A node

```
msg = tsend(peer,msg,L) ;
msg = trecv(peer) ;
compute(comp_time) ;
```

Class B node

```
msg = trecv(peer) ;
msg = tsend(peer,msg,L) ;
compute(comp_time) ;
```

Base Loop

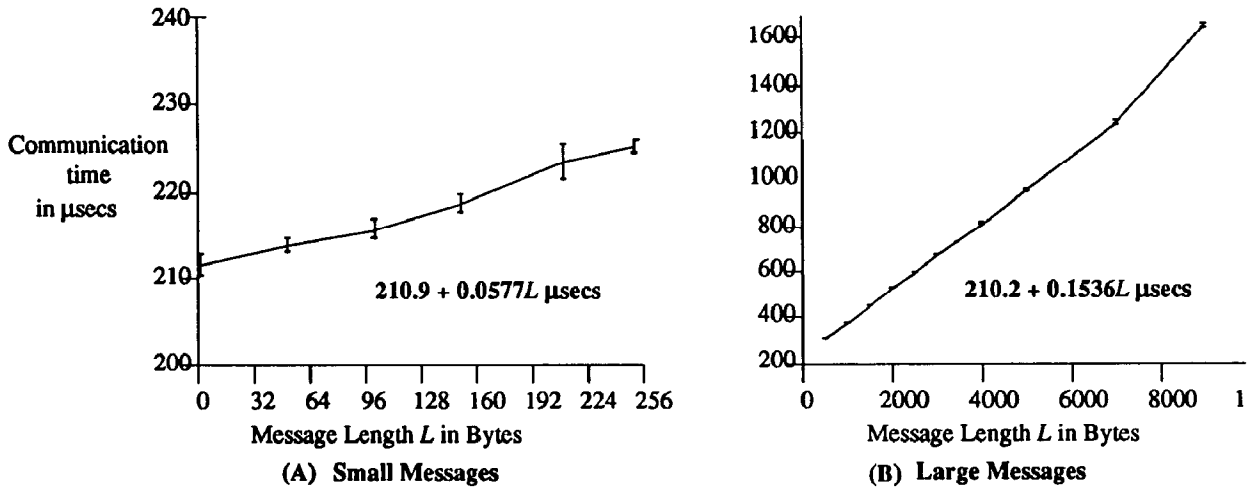
Figure 3 : Contention-free Adjacent-node Communication

on each network channel, the traffic is only along one direction at any instant. In addition, both the peers compute for the same amount of time after completing their communication. This strategy ensures that the receiving node will always be ready for the incoming message.

The communication time per loop can be computed by subtracting *comp_time* from *loop_time*. The communication time will not change as the computation time changes due to the reasons mentioned in the previous paragraph. The communication time is equal to twice the time to communicate a single message of length L between adjacent nodes. Thus, the time to communicate a single message can be computed from the average *loop_time*. The experiment was repeated for different values of message lengths (L), and communication time per message was plotted as a function of the message length in Figure 4. The plot shows the mean of the values computed by 16 nodes. Even though different pairs could progress at their own rates, we do not see any significant difference in the average loop time. The variation in loop_times over different nodes is less than 4 μsecs for messages up to 8Kb long. This small variation demonstrates that the combined performance of the hardware and the kernel of individual nodes is almost identical.

These experiments show that the communication time per message can be specified as a linear function of message length L, i.e. $\alpha+\beta L$. Here, α is the fixed overhead and β is the effective bandwidth. We observe that fixed overheads are still significant in this second generation machine - about 210 μsecs. Our measured fixed overhead is slightly larger than the value of 177 μsecs reported recently [22]. The effective bandwidth is 1/0.0577 ≈ 17 Mbytes/sec for small messages (≤ 256 bytes) which is about 87% of the channel bandwidth of 20 Mbytes/sec. Larger messages are sent as packets of 256 bytes, and we observe in Figure 4B that packetization has a noticeable impact on the effective bandwidth. The packetization overheads will reduce the effective bandwidth to 1/0.1536 ≈ 6.5 Mbytes/sec. Long messages, therefore, proceed about 2.7 times slower than smaller messages. The effective bandwidth is again slightly lower than the claimed value of 1/0.11 ≈ 9 Mbytes/sec [15]. The packetization is enforced to prevent long messages from blocking the transfer of small messages; there is no way to avoid this packetization in the current system.

3.2 Contention-free Non-Adjacent node Communication : In this section, we examine the performance for communication between nodes that are not physically adjacent. Contention for both rr-channels and pr-channels is once again avoided so that we can concentrate on the effect of longer path lengths. We achieve that goal by only allowing nodes along the the leading diagonal to be active, that is node (i,j) belongs to class I (inactive) if $i \neq j$. A leading diagonal node belongs to class A if $i < \sqrt{N}/2$, and to class B, otherwise. The peer of node (i,i) is node $(\sqrt{N}-i-1,\sqrt{N}-i-1)$, so a node and its peer belong to different classes. Thus, we have $\sqrt{N}/2$ pairs communicating - avoiding any interference [Figure 5]. The result is no contention for rr-channels (network channels). The base loop is the same as shown in Figure 3. Hence, contention for pr_channels is avoided as before.



(A) Small Messages **(B) Large Messages**

Figure 4. Time for Contention-free Adjacent-node Communication

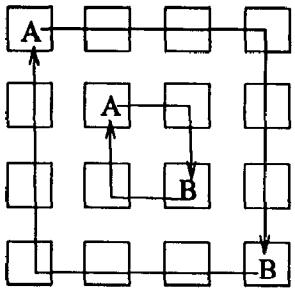


Figure 5. Contention-free Non-adjacent node communication

The path lengths for the various pairs are 2,4,6,...,$2\sqrt{N}$-2. If longer path lengths were to increase delays, we would expect to see communication time (and hence loop time) increase as we move from node ($\sqrt{N}/2$,$\sqrt{N}/2$) to node (0,0). The average loop times for the diagonal elements of a 12 × 12 2D mesh are shown in Figure 6. The times are for 4Kb messages and 480 μsecs average computation time per loop (uniformly distributed in the interval 0..960 μsecs). The times in Figure 6 do not show any dependency on path length. Figure 7 shows how time for a single message varies with path length for different message sizes. The change is less than ±4 μsecs. Since such variations existed even for adjacent node communication, we conclude that the effect of path lengths can be ignored in current wormhole-routed networks. The times here are slightly different than the values given in Section 3.1, as the systems and CE kernels were different.

3.3. Communication under Contention : In this section, we describe several experiments designed to study the effect of contention on the communication time. We establish communication patterns so messages will have to compete for network channels (rr_channels) and/or P.E. to router channels (pr_channels). Unlike the experiments in the previous two sections, the computation time will affect the communication time by changing the probability of contention. In the experiments described in this section, the computation time per loop is uniformly distributed over the interval 0..$2t_a$, where t_a is the mean value.

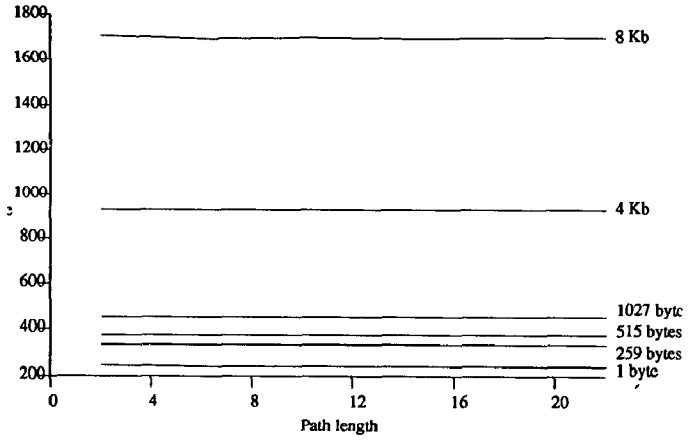


Figure 7. Communication time per message vs Path length.

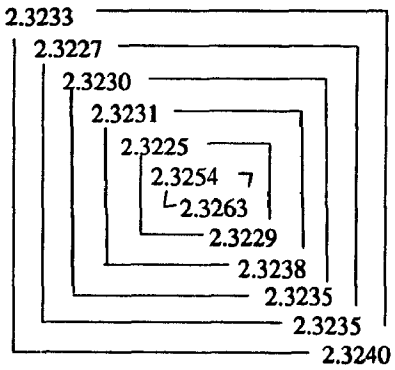
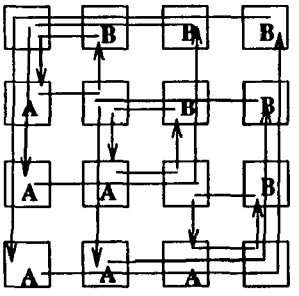


Figure 6. Average loop time for the leading diagonal nodes in μsecs

3.3.1 Contention for Network Channels : In order to study the effect of contention for network channels, we need a communication pattern such that the nodes are affected differently by contention and that the nodes progress independently of each other. Under these conditions, it is possible to measure the effect of contention by comparing the difference in average loop times. A pattern that satisfies those conditions is what we call the *matrix transpose pattern*. In this pattern, all nodes along the leading diagonal are idle, i.e. node (i,j) belongs to the class I only if $i = j$. Any other node belongs to class A, if $i > j$, or belongs to class B otherwise. As in the previous experiments, each node has a peer with which it communicates and peer (node (i,j)) is node (j,i). Thus, we have $(\sqrt{N}(\sqrt{N}-1))/2$ pairs communicating and, unlike the previous experiments, their communication results in contention [Figure 8]. The base loop is identical to that of the experiments in 3.1 (Figure 3). Hence, from the node's point of view, it is performing the same task as in the previous experiments. However, the experiment is more demanding on the network because the communicating nodes are widely separated, and the messages between them will result in contention for network channels. Any contention for the pr-channel is avoided as before.

An examination of Figure 8 provides an idea of the distribution of contention due to communication in the matrix-transpose pattern. To construct that contention pattern, we have utilized the fact that Symult 2010 systems use a fixed routing scheme. Messages travel along rows first, and then along the columns. The channels connected to the two corner nodes (0,0) and ($\sqrt{N}$-1,$\sqrt{N}$-1) have the maximum loads, whereas the channels connected to the other two corner nodes have the lowest loads. The loads on other channels vary between those two extremes. When one considers the effect of such channel loads on individual nodes, we find a wide distribution of effects. Nodes that are furthest from the leading diagonal are affected the most. Thus, it is the corner nodes ($\sqrt{N}$-1,0) and (0,$\sqrt{N}$-1) that are affected most by the contention. These two nodes are peers, and messages between them traverse the longest possible paths. A message from node (0,$\sqrt{N}$-1) to its peer node ($\sqrt{N}$-1,0) has to compete for each of the $\sqrt{N}-1$ horizontal segments (channels) of its path. On the other hand, a node near the principal diagonal, such as node (0,1), only will have to compete for the one channel between nodes (0,0) and (0,1). Thus, the average loop time should



Communication Pattern
(16 nodes)

--, 20847, 20919, 20955, 21020, 21090, 21158, 21096
20840, --, 20861, 20994, 21046, 21160, 21166, 21151
20919, 20857, --, 20947, 21009, 21059, 21118, 21103
20955, 20991, 20939, --, 20925, 20992, 21041, 21027
21018, 21040, 21009, 20922, --, 20912, 20925, 20920
21087, 21158, 21053, 20983, 20907, --, 20880, 20856
21146, 21161, 21121, 21033, 20917, 20878, --, 20863
21094, 21144, 21098, 21021, 20919, 20851, 20865, --

Time for 10000 loops in ms
(64 nodes)

Figure 8. Matrix-Transpose Communication

increase for nodes that are farther away from the diagonal, if the increase in delay due to contention is significant.

The loop times for a specific case when the message length is 4Kb and the mean computation time is 250 μsecs is shown in Figure 8. The completion times clearly increase as we move away from the principal diagonal. Let t_{min}, t_{avg} and t_{max} denote the minimum, average and maximum loop time taken by any node in this experiment. Let t_{min}', t_{avg}' and t_{max}' be the corresponding values when the peers are mapped onto adjacent nodes as in the experiment in Section 3.1 ($t_{min}' \approx t_{avg}' \approx t_{max}'$). The increase in loop time (and hence in communication time per loop) for an average and a worst-affected node (due to contention) are plotted for various systems sizes in Figure 9. The solid lines indicate the increase in communication time for an average node, i.e. $t_{avg} - t_{avg}'$. The dotted lines indicate the increase for the worst affected node, i.e. $t_{max} - t_{max}'$. Note that *decreasing* the average computation time per loop while keeping the number of communication steps fixed, has the effect of *increasing* the ratio of communication time to computation time. Also, observe that the y-axis of Figure 9 increases exponentially. Our results in Figure 9 for large messages show that the communication time increases exponentially both with the communication-to-computation time ratio and the system size.

Since we showed in Section 3.2 that path lengths cannot account for that exponential increase in communication time we find here, we conclude that the increase is due to contention for network channels. Although these results show that the increase in communication time is significant only for large messages and very high communication to computation ratios, we show in the next section that these delays can occur for smaller messages and more reasonable communication to computation ratios.

3.3.2 Contention for P.E. to router channels : We now examine the effect of contention for the P.E. to router channel, the pr-channel. Contention for pr-channels will occur if a node receives and/or sends several messages simultaneously. In fact, multiple, simultaneous sends and receives will also result in contention for any node hardware involved in sending or receiving a message, including the CPU. Such contention was carefully avoided in all our previous experiments. Contention for pr-channels is more serious in current systems than contention for network channels. For many common communication patterns, contention for pr-channels can occur causing the communication times to increase.

The architecture of a node (shown in Figure 1) indicates the problem. All five channels (four rr-channels and one pr-channel) have the same bandwidth. If messages arrive on more than one rr-channel to a given node, then the messages have to compete for the single pr-channel and create a bottleneck. This pr-channel bottleneck can slow down an incoming message by a maximum factor of eight. We use variants of the popular near-neighbor communication patterns here to demonstrate the effects of contention for pr-channels. The communication patterns are shown in Figure 10.

In near-neighbor patterns, a node typically exchanges data with one or more neighbors in each loop. Consider the experiment in Section 3.1. In that experiment, each node exchanged (sent and received) a message with its peer in each loop. Contention was avoided for both rr-channels and pr-channels. Let us call that pattern the *AB−1NN* pattern: one Near Neighbor and two different classes of nodes, *A* and *B*. The base loop is the same as given in Figure 3. If we now make all nodes belong to the class *A*, then each node will first initiate a send and then receive the incoming message from its peer [Figure 10]. We call this the AA-1NN pattern (all nodes in Class A). In this pattern, a node will be simultaneously sending and receiving. Traffic exists in both directions on pr-channels as well as rr-channels. Since messages are progressing concurrently, we expect the communication time per loop to be half of that for the *AB−1NN* pattern. However, the measured times indicate that it is almost the same as for a *AB−1NN* pattern [Figure 12]. Thus, even though all the messages are progressing concurrently, the communication time is twice that for a single message under contention-free conditions. In other words, if traffic exists in both directions, the communication time for a message is increased by a factor of 2. This increase is due to the contention for the pr-channel which has become the bottleneck.

Instead of sending and receiving from one neighbor, we next consider the case where a node communicates to all its four neighbors in each loop - a common situation. Often, a node sends the same message to all the four neighbors in each loop. The four-neighbor pattern which avoids contention for network channels is given in Figure 10 (AB-4NN pattern). The base loop of A and B nodes is given in Figure 11. A node (i,j) belongs to class *A* (*B*) if $i+j$ is even (odd). Even though contention for network channels is avoided, four messages (copies of one) will be contending for the pr-channel. We see that even for long messages, the communication time for the

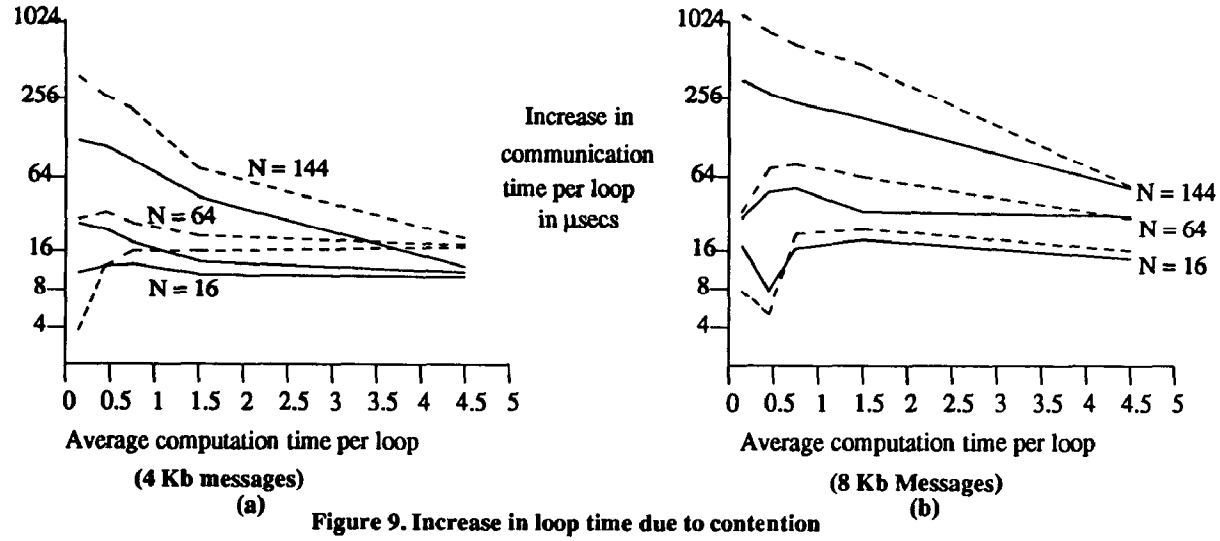


Figure 9. Increase in loop time due to contention for network channels (Matrix_transpose pattern)

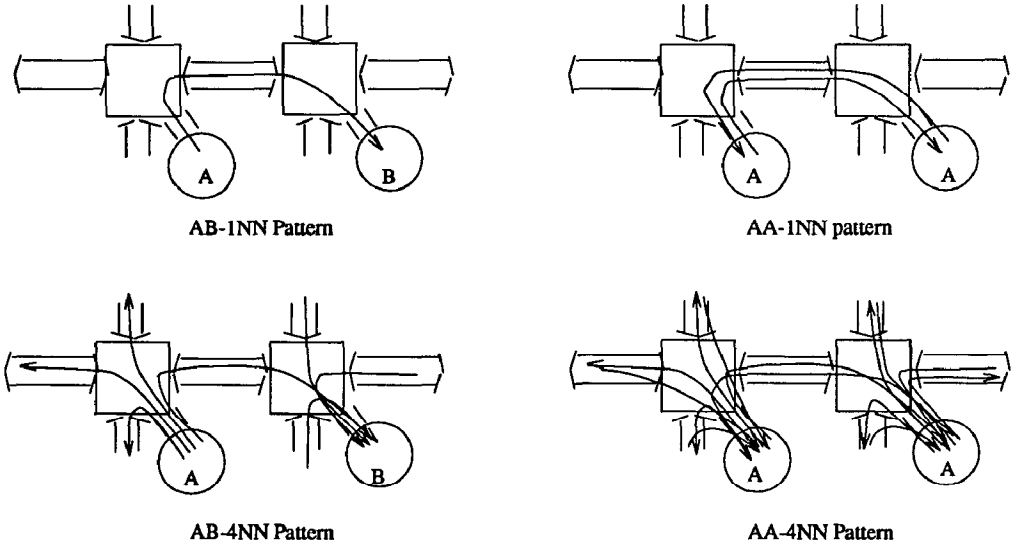


Figure 10. Near-neighbor Communication Patterns that result in multiple sends and receives

AB-4NN pattern is four times that of the AB-1NN pattern. Hence, the communication time per loop is eight times that for a single message. As in case of the 1NN pattern, we can place all nodes in class *A* to create the AA-4NN pattern. The loop times for AA-4NN pattern are in Figure 12. Again, we find little difference in the loop times even though the AA-4NN pattern initiated all sends first before initiating any receives. As before, the problem is due to the contention for the pr-channel by the four incoming and the four outgoing messages.

The experiments in this section lead us to conclude that the pr-channel can become a serious performance bottleneck in current systems.

4. Performance Analysis

In Section 3.3.1, we showed that contention for network channels can increase communication times exponentially. However, our experiments show that the increase in communication time is significant (>10 μsecs) only for messages larger than 1 Kb. In this section, we analyze the results further and show that the increase in delays can be significant even for smaller messages and higher computation times per loop. Figure 9 shows the absolute increase in loop time (and hence communication time per loop) in μsecs for a given value of computation time. Our goal is to convert those graphs to relative measures that avoid any dependency on machine parameters, and better characterize the effect of contention. We will show that the percentage increase in communication time is an exponential function of the utilization level of the channels.

First, let us look at the components of *loop_time* for contention-free communication. Figure 13, shows the

A Node	*B Node*
for each neighbor **do**	**for** each neighbor **do**
tsend (neighbor,msg,msgln) ;	msg = trecv (neighbor) ;
for each neighbor **do**	**for** each neighbor **do**
msg = trecv (neighbor) ;	tsend (neighbor,msg,msgln) ;
compute (*comp_time*)	compute (*comp_time*) ;

Figure 11. Base loop for four near-neighbor (4NN) pattern

Msg ln	AB-1NN		AA-1NN		AB-4NN		AA-4NN	
(in bytes)	t_1	t_1/t	t_2	t_2/t	t_3	t_3/t	t_4	t_4/t
1	.412	2.00	.385	1.87	2.422	11.75	2.269	11.01
515	.687	2.00	.681	1.98	3.425	9.97	3.059	8.90
1027	.791	2.00	.785	1.98	3.782	9.56	3.683	9.31
4099	1.767	2.00	1.766	1.99	7.648	8.65	7.508	8.49
8196	3.306	2.00	3.295	1.99	12.938	7.82	12.793	7.73

Figure 12. Communication time per loop for near-neighbor patterns
(times t_i are in msecs. t is the time for a single message as given in 3.1)

components for a specific instance, when the message length is 4 Kb and the average computation time per loop is 1 ms. The communication time per loop includes time to send a 4 Kb message from an *A* node to its peer (A-B communication) and time for another 4 Kb message from the peer back to node *A* (B-A communication). Each message was shown in Section 3.1 to take 210.2+0.1536 × 4092 = 0.839 ms. Of this, 0.21 ms is fixed overhead and 0.629 ms is needed to send the 16 packets of 256 bytes each. Network channels need only 0.0577*256 = 14.8 μsecs per packet - shown as shaded regions in Figure 13. The remaining time, (0.1536 - 0.0577)*256 = 24.6 μsecs is overhead to process each packet.

Overheads and other components of loop time may differ in future systems for the same message length and amount of computation per loop or, in other words, the same base loop. For example, packetization may not be required, which would allow elimination of packetization overheads. Also, fixed overheads may be reduced. Given those improvements we would like to be able predict the increase in communication times for a given base loop. The three important parameters of the base loop are

Message length L : the number of bytes sent over a path between two nodes per loop.

Physical Communication Time t : the time taken by the network channels to transmit *L* bytes = L/B, where *B* is the average communication rate of a network channel.

Loop time T : the time for one loop under contention-free communications. It can be computed if we know the base loop. For example, consider the base loop given in Figure 3

$$T = \text{comp\_time} + 2(\text{comm\_time}) = \text{comp\_time} + 2(\alpha+\beta L)$$

where $\alpha+\beta L$ is the time to send a message of length *L* bytes.

The α and β values can be easily determined for a given machine by the methods of Section 3.1. With these values, *T*, which gives the loop time under contention-free conditions, can be computed.

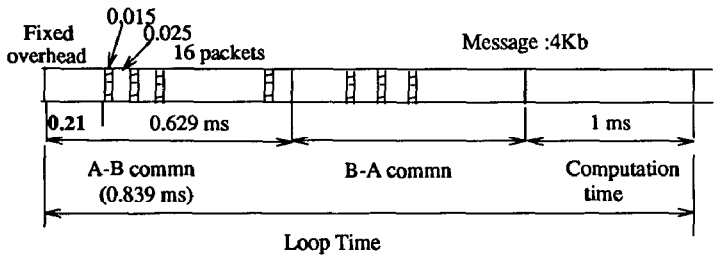


Figure 13. Components of Loop time

Given these parameters we will define the concept of utilization in this environment. Observe that *t* is contained in the βL component of *T* and that $\beta L \geq t$. We know that contention increases the physical communication time from *t* to $t+\Delta t$. That increase in communication time will increase the loop time to $T' = T+2\Delta t$, where T' is the loop time in the presence of contention. Furthermore we expect that the percentage increase in physical communication time ($\Delta t/t$) to be a function of the utilization level of the channels. The utilization of an entire channel can be expressed as pt/T', where *p* is the number of paths sharing the channel and t/T' is the percentage of time bits are crossing the channel for each path. The value of *p* for any channel is fixed for a given communication pattern. We can further refine the concept of utilization by defining the *path utilization* as t/T, the fraction of time a path keeps a channel busy under contention-free conditions. Using the results of the experiments in 3.3.1 for 4 Kb messages we plot this function for various system sizes in Figure 14A. Figure 14B shows that delay for other message lengths have the same shape indicating that these curves could represent a function that gives the percentage increase in communication times for a given path utilization.

The curves of Figure 14 are useful in several ways. First they show that an increase in communication time due to contention is an exponential function of path utilization (or channel utilization). Observe the exponential scale on the y-axis. The curves can be used to compute the increase in delays due to contention. Since *t* and *T* can be easily computed, we can find the path utilization t/T, and use Figure 14 to find the percentage increase in the communication times, $\Delta t/t$. Using that value we can compute Δt. The increase in loop times can also be calculated; it is $2\Delta t$ for our base loop of Figure 3.

Figure 14B also shows why we expect the increase in delays to be significant in future systems even for smaller messages. If overheads are reduced, path utilization increases for the same message length and computation time. The increased path utilization will increase the delays due to contention. In fact, with current overhead levels and architectural bottlenecks we could never increase path utilization beyond 7% for 1K byte messages and 12% for even 8K Byte messages. If those architectural impediments are removed in future systems, we expect increased path utilization even for smaller messages.

Another inference we can derive from the curves in Figure 14A concerns the effect of an increase in system size. Each node is still executing the same base loop, so path utilization remains the same, but communication times will increase. This increase is due to the non-uniform loads on the

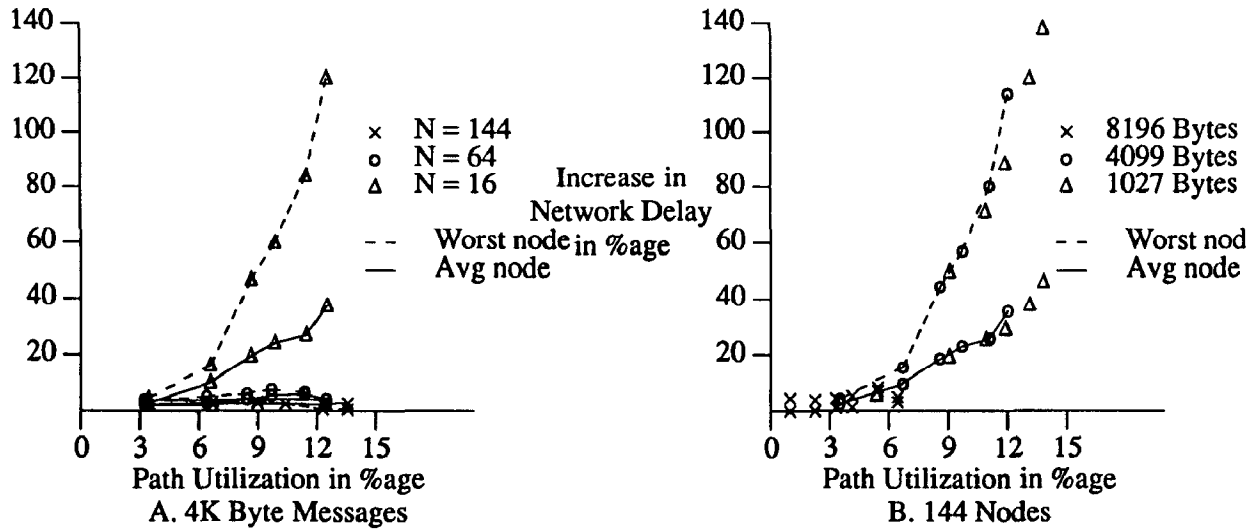


Figure 14. Increase in Network Delay vs. Path utilization

channels (Figure 8). The channel between (0,0) and (0,1) has the highest load as $(\sqrt{N}-1)$ paths are sharing the channel. This utilization is approximately $(\sqrt{N}-1)$ times the path utilization. For a 144 node system $\sqrt{N}-1 = 11$ so even a 12% path utilization on each of those 11 paths will saturate the channel. We, therefore, find that even though path utilization remains the same, channel utilization will increase with system size. Also, as explained in Section 3.3.1 messages must compete for more channels as system size, and hence path lengths, increase. The result is that communication time increases dramatically as system sizes increase. That trend is apparent in Figure 14A.

5. Discussion

In the previous section we saw that, although path lengths are not a problem in current wormhole-routed networks, contention can be a serious problem. The Symult 2010 systems used in our experiments may not be typical representatives of the class of 2D mesh connected, wormhole-routed networks. In this section we discuss some of the architectural idiosyncrasies and drawbacks of the Symult 2010 system which may not be present in future systems, and explain the effect of those characteristics on our conclusions.

3.1 Router-to-Processor channel : In a Symult 2010, the router-to-processor channel (pr-channel) bandwidth is the same as as that of the channels between nodes. Hence, the pr_channel has become a bottleneck for several communication patterns and limits the peak rates at which a node can inject traffic into the network. It is possible to increase the bandwidth between the router and the processor by either increasing the channel width or the channel rates. Ideally 4 times the bandwidth of the network channels in each direction should be sufficient to eliminate this bottleneck. In addition, the node and its kernel should be designed to handle multiple incoming and/or outgoing messages. Future designs, such as iWarp [2], are increasing the bandwidths between the communication and the computation processor. These improvements, however, will increase the path utilization and the network traffic and hence the problem of contention for network channels.

3.2 Packetization Overheads : In a Symult 2010, messages larger than 256 bytes are sent as 256 byte packets. This was done to prevent large messages from affecting the communication of smaller messages due to contention for channels. However, this introduces additional software overhead which causes larger messages to progress 2.7 times slower than smaller messages. In addition, a user cannot prevent packetization, even if he knows no contention exists. Our results indicate that turning off packetization while avoiding contention could yield a significant performance improvement.

3.3 Multicast and Broadcast support: At present, support for multicast and broadcast communication is inefficient - multiple sends are performed. This slows down the broadcast by a factor of N. Architectural support for broadcast and multicast will enable a node to inject greater traffic into the network. This increased traffic will increase the contention problems.

3.4 System Size : At present, the largest Symult system has 192 nodes. However, system sizes are expected to be as high as 1K nodes or more in the near future. As we saw earlier, the utilization of some channels can increase due to non-uniform traffic to very high levels even though a node generates messages at reasonable rates as measured by the metric *path utilization*. Path utilization is a characteristic of the parallel program and remains the same irrespective of the system size. Thus, contention and channel utilization will scale up with the system size because more paths will share a channel. Also, longer path lengths will force a message to compete for more channels along the path increasing contention.

3.4 Fixed Software Overheads : In current systems, the fixed overheads per message is about 210 μsecs which is quite high compared to network delays. It is possible to reduce these overheads. as is being done in the new iWarp design [2]. The result is efficient communication, even for smaller messages as messages can be injected at a faster rate. In such a system, contention will affect smaller messages.

3.5 Wider Data paths : Future systems are being designed to have wider data paths. This could be the main factor to reduce

the effect of contention. As the width of the channels increases, the network delays become smaller which will reduce the path utilization dramatically. The increased width decreases the message aspect ratio (the ratio of message length to the width of the network channel) to the point where it is probably comparable to the network dimension. The effect of contention again will be complex due to blocked messages tying up the network resources, i.e. channels.

6. Conclusion and Future Research

The results of our experiments and analysis of these results lead us to conclude the following.

1. The communication time per message under contention-free conditions is one of the best any implementation has achieved, and is given by the linear function 210.9 + 0.0577L μsecs (or 210.2 + 0.1536L μsecs if $L > 256$) where L is the message length. This expression is accurate to within ±4 μsecs, even for paths of length 22.

2. Contention for network channels increases the communication time exponentially as the communication-to-computation ratio increases. In current systems, the increase is significant (> 10 μsecs) only in the largest system (144 nodes) and only for long (> 1K byte) messages. Small systems sizes and large fixed overheads are limiting the load on the network and the effect of contention in current machines.

3. Improvements are needed in the current architecture to reduce fixed overheads (setup times), and to provide efficient broadcast/multicast support.

4. Although the improvements mentioned in item 3 will improve the ICN performance for contention-free communication, those improvements will increase the effect of contention for network resources. The result will be that contention will become a problem for smaller messages.

A number of research issues are suggested by this work. One area of research would be to improve some of the weakness of the ICN architecture mentioned above -- the iWarp [2] is doing that. Another research area is the effect of contention and random mapping in systems that are larger than currently available and in systems with reduced overheads. We are currently studying those issues. A final research area is a reconsideration of the mapping problem based on contention rather than path length.

Acknowledgement : We thank Prof. Seitz at Caltech for allowing us to use Caltech's Symult 2010 systems for our experiments.

References

[1] Luc Bomans, Dirk Roose, "Benchmarking the iPSC/2 hypercube multiprocessor", *Concurrency : Practice and Experience*, Sept 89, pp 3-18.

[2] S. Borkar, et al, "iWarp : An Integrated Solution to High-speed Parallel Computing", *Supercomputing Conference*, pp. 330-339, 1988.

[3] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D.Reed, "Hyperswitch Network for the Hypercube Computer", *International Conference on Parallel Processing*, pp. 90-99, 1988. 1987, pp 224-232.

[4] W.J. Dally, P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller", *Proc IEEE International Conference on Computer Design*, pp. 230-234, 1987.

[5] W.J. Dally, "Wire Efficient VLSI Multiprocessor Communication Networks", *Proceedings Stanford Conference on Advanced Research in VLSI*, pp. 391-415, MIT Press, Cambridge, Mar 1987.

[6] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks", *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775-785, June, 1990.

[7] C. Flaig, "VLSI Mesh Routing System", Dept of Computer Science, California Institute of Technology, Technical Report 5241. May 1987.

[8] D.C. Grunwald, D.A. Reed, "Benchmarking Hypercube Hardware and Software", *Third Conference on Hypercube Concurrent Computers and Applications*, pp. 169-177, 1987.

[9] D.C. Grunwald, D.A. Reed, "Networks for Parallel Processors: Measurements and Prognostications", *Third Conference on Hypercube Concurrent Computers and Applications*, pp. 610-619, 1988.

[10] Kermani, Parviz and Leonard Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks* , vol. 3, pp. 267-286, 1979.

[11] S.F. Nugent, "The iPSC/2 Direct-Connect Communication Technology", *Third Conference on Hypercube Concurrent Computers and Applications*, pp. 51-60, 1988.

[12] A.G. Ranade, S.L. Johnsson, "The Communication Efficiency of Meshes, Boolean cubes and Cube connected cycles for Wafer Scale Integration.", *International Conference On Parallel Processing*, pp. 479-482, 1987.

[13] D.A. Reed, D.C. Grunwald, "The performance of Multicomputer Interconnection Networks", *IEEE Computer*, pp. 63-73, June 1987.

[14] C. L. Seitz, "The Cosmic Cube", *Comm of ACM*, Vol 28, No 1, Jan 85, pp 22-33.

[15] Charles L. Seitz, Jakov Seizovic, Wen-King Su, "The C Programming's Abbreviated Guide to Multicomputer Programming", Technical Report, Caltech-CS-TR-88-1, California Institute Of Technology, Pasadena, April 1989.

[16] C.L. Seitz, et al, "The Architecture and Programming of the Ametek Series 2010 Multicomputer", *Hypercube Computers Conference*, pp. 33-36, 1988.

[17] P.Y. Song, "Design of a Network For Concurrent Message Passing Systems", *M.S. Thesis*, pp. 1-119, Dept of EECS, MIT, May 1988.