Execution Based Evaluation of MINs for Cache-Coherent Multiprocessors *

Akhilesh Kumar, Laxmi N. Bhuyan and Ravishankar R. Iyer
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA.
E-mail: {akhil,bhuyan,ravi}@cs.tamu.edu
Phone: 409-845-5007

Abstract

In this paper, performance of multistage interconnection network with wormhole routing and packet switching has been evaluated for cache-coherent shared-memory multiprocessors. The evaluation is based on execution-driven simulation using various applications. The traffic in cache-coherent systems is very different from the traffic in message-passing environment and is characterized by traffic bursts, one-to-many and many-to-one traffic, and small fixed length messages. The performance of packet switching and wormhole routing has been evaluated for different buffer sizes. The comparison of execution time between packet-switched and wormhole-routed network for same amount of buffer space per switch shows that wormhole-routed network provides significant advantage over packet-switched network. We have also evaluated wormhole networks with virtual channels for variable number of virtual channels and flit buffers per channel. The study shows that for wormhole routing 4 virtual channels per link and 4 flit buffers per channel is the best configuration in most of the cases.

Keywords: Interconnection network, wormhole routing, packet switching, virtual channels, performance evaluation, shared-memory multiprocessor.

*This research has been supported by NSF grant MIP-9301959.

1 Introduction

Large-scale shared-memory multiprocessors are difficult to design but they provide a unified view of the memory for easy programming. These systems are built using processor-memory nodes that are connected through an interconnection network (IN), such as hypercube, mesh, multistage network, etc., in a distributed shared memory organization. Cache memories are an integral part of such systems to avoid the large latency of remote memory accesses. The time to service a cache miss from a remote memory in a large system could be several orders of magnitude higher than the cache access time. Thus it is desirable that the IN provides minimum latency for servicing the cache misses.

The presence of cache requires support for maintaining coherence among copies of data present in the caches and memory. A lot of research has been done in developing cache coherence protocols for large-scale systems [1]. The focus of this paper is on the evaluation of the IN in cache-coherent shared-memory systems for different applications through execution-driven simulation. The performance evaluation of multiprocessor interconnection networks has been an active area of research for a long time [2]. This has contributed to the advances in the design and implementation of the networks to a great extent. However, these advances have made the networks much more complex and it is difficult to capture all the details of the network into a simple analytical or simulation model. Moreover, the effect of all these advances in the INs has to be judged from the real changes in the execution time of scientific applications. Therefore, we have chosen the approach of execution-driven simulation to capture as much detail as possible.

Circuit switching, packet switching and wormhole routing [3] are the switching techniques used by INs in multiprocessors. In this paper we will consider only the packet switching and wormhole routing techniques because circuit switching is known to be less efficient. In case of packet switching, a message is divided into fixed size packets, each packet having its own header for routing information. The packets are moved from one switch to another as a unit. Wormhole routing is an efficient technique to reduce the latency of internode messages. In wormhole routing, the messages are divided into small flits, typically 8-32 bits long, and sent over the network in a pipeline fashion, one flit after another between adjacent switches.

The performance of MINs for packet switching and wormhole networks has been evaluated in various studies [4, 5, 6, 7, 8], based on analytical models that assume simple traffic distributions. These workload models are far from the network traffic observed in a cache-coherent shared-memory multiprocessor, which is characterized by invalidation messages, bursty requests, memory hot-spots, and different message sizes. Moreover, these models evaluate the network in terms of its throughput and message delay, instead of response times of applications. An analysis of the packet-switched MIN with directory-based cache-coherence protocol was presented in [5] under the uniform memory access assumption.

Therefore, our aim is to study the performance of the network with the execution of real applications and assess the impact of various network parameters and switching techniques.

In a related work [9], the performance of the multistage interconnection network in Cedar multiprocessor was studied through measurement using several applications. The authors were able to identify bottlenecks in the network due to traffic convergence and studied the effect of several options to avoid congestion. But, the Cedar multiprocessor, not being a cache-coherent multiprocessor, the study does not indicate how the presence of caches affects the performance of an IN.

Virtual channels [4] are used in wormhole networks to avoid deadlocks and to improve link utilization and network throughput. Therefore, we have also evaluated the performance of a wormhole-routed MIN together with virtual channel flow control in a cache-coherent multiprocessor. The evaluation has been done by varying the number of virtual channels and flit buffers per virtual channel.

In this paper, we evaluate the performance using an execution-driven simulation where traffic is generated using real applications. The work provides insight into the bottlenecks and hot-spots in the network due to cache coherence traffic. Here we would like to highlight the capabilities of our simulation model which is difficult to incorporate in any analytical model due to their complexity. Apart from doing an accurate evaluation of buffered packet switching, wormhole routing and virtual channels with proper blocking of packets(flits), we can experiment with various message scheduling policies to resolve contention among multiple messages trying to acquire the same channel, link allocation policies among virtual channels, buffer sizes, and memory management policies. Also, we implement a directory-based cache-coherence protocol [10] in our simulator and realistic synchronization techniques [11].

Consideration of all the network parameters and their effect on the system performance is beyond the scope of this paper because of the time and space constraints. Here, we will limit ourselves to two switching techniques – packet switching and wormhole routing. The comparison of packet switching and wormhole routing will be done for fixed buffer space per switch. In case of wormhole routing we will also consider the impact of virtual channels. The characterization of traffic in cache-coherent systems and the effect of message scheduling and link allocation policies are being studied at present.

The following section presents the system model and the coherence protocol implemented for the simulation. Section 3 discusses the relevant features of the applications used in the evaluation. Section 4 presents and discusses the average message latencies observed for different network architectures. Section 5 presents the execution times of each application for different network architecture. Finally, section 6 presents the conclusion and the direction for further work in this area.

2 Simulation Model

Our simulator is based on PROTEUS [12]. However, the simulator implemented MIN using an analytical model presented in [13]. We have modified the simulator extensively to exactly model MINs with packet switching and wormhole routing. For wormhole routing, we have also incorporated virtual channels and multi-flit buffers. The system considered for evaluation in this paper is a directory-based cache-coherent shared-memory multiprocessor connected through a MIN using 2×2 switches. In this section we will describe the model of the network interface, the switch, and the cache coherence protocol. In the rest of the paper we refer to physical connections between switches as **links**. Virtual connections or the set of buffers belonging to a virtual channel are referred as **channels**.

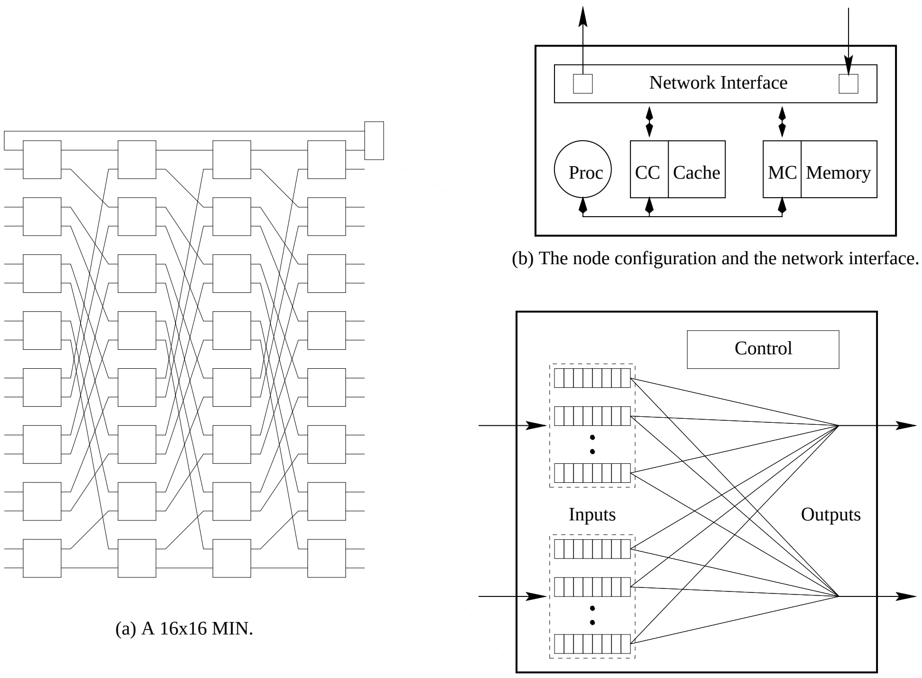
2.1 The network interface and the switch

The MIN structure used in our simulation model is shown in Fig 1(a). The processing nodes in the system consist of processor, cache, cache controller, a section of the distributed shared-memory including the memory controller, and a network interface, as shown in Fig 1(b). The nodes are connected to a switch through the network interface. The input and output links of a switch connect to other switches to form the multistage IN. The system implements a full-map directory-based cache-coherence scheme [10] which will be described later.

The processors are assumed to have only a single thread and no prefetching of cache blocks is performed. The system is assumed to be sequentially consistent, which means that there are no write buffers and cache misses on load or store operations block the processor. This implies that there is only one outstanding memory request from a processor at any time. However, this does not mean that there is only one outstanding message from each node. Messages are also generated by the cache and memory controller in response to remote requests and coherence actions, which are independent of the state of the processor.

In the results presented in this paper we have considered that the shared memory is distributed such that high-order bits of the address represents the node where it is located. So, the adjacent blocks of memory are located at one node. The network interface, Fig 1(b), provides buffer space for all incoming and outgoing messages. We assume that network interface has enough space to store all messages. The interface also provides services such as dividing the message into packets or flits, initializing header flit or every packet with necessary routing information, etc.

The switches contain limited amount of buffers at each input to store the incoming messages. This buffer space is organized differently based on the network architecture. For packet switching, the buffer space is organized in terms of fixed size packets. For wormhole routing, it is organized as a set of flits. In case of wormhole routing with virtual channels,



(a) A 16x16 MIN.

(b) The node configuration and the network interface.

(c) Wormhole router with virtual channels

Figure 1: The system architecture.

the buffers at each input is divided into a set of virtual channels, as shown in Fig 1(c). Each of the virtual channels can have space for multiple flits. Flits from only one message can be occupying the buffers at one virtual channel at a time. There are no buffers on the output side to avoid an unnecessary cycle to copy a flit to an output buffer before forwarding it to the next switch. The routing decisions and flit movement are done using the following steps during every clock cycle.

1. Find *free* and *ready* output channels (same as the input channel of the next node in the path). A channel is *free* if it is currently not assigned to any message, i.e., all its buffers are empty. A channel is *ready* if its buffers are not full and it can accept a flit. A *free* channel is also *ready*, but not vice-versa.

2. For the new messages (header flit), the controller requests for a *free* output channel depending on the routing function. If the routing function allows multiple output channels, then a request is made to only one of the *free* output channels at a time.

3. If multiple new messages request for the same output channel, then a message selection policy is used to resolve conflicts. In this paper, we assume a simple message selection policy that assigns an output channel to input channels in a round-robin fashion. This ensures equal priority to all the input channels and avoids starvation of messages using any particular channel. If a message fails to get an output channel, it tries again in

the next cycle.

4. Link allocation is done among *ready* output channels that have a flit to send to next node on the path. Among these channels, the link is allocated in a round-robin fashion, however, the channel that used the link during the previous cycle is considered first. This scheme provides equal priority to all the channels and keeps the average message latency small.

In case of packet switching, the messages are divided into fixed size packets, with each packet containing the routing information. The packet is transferred from one switch to another as a whole. A packet is forwarded to the switch in the next stage only if it has enough space to store the full packet. In case of contention at the output link, it is resolved in a round-robin fashion. We have not modeled split buffers at the inputs of the switch.

2.2 Cache coherence protocol and synchronization

We implemented the full-map directory-based cache coherence protocol [10] for evaluation in this paper. In this scheme, each shared memory block is assigned to a node, called *home node*, which maintains the directory entries for that block. Each entry in the directory is a bit-vector of same length as the number of nodes. The directory also maintains the information about the state of the blocks. Whenever a copy of a memory block is sent to a cache, the bit corresponding to that node is set. An invalidation protocol has been implemented in which all the cached copies of a block are invalidated on a write operation. The system is assumed to be sequentially consistent, meaning that there are no write buffers and cache misses on load or store operations block the processor.

The protocol has been modified to make it work in a network that does not guarantee in-order delivery of messages between a source - destination pair. In a network with virtual channels, it is possible that messages may arrive out-of-order at the destination from the same source. When a message gets blocked in the network, it is possible that another message from the same source to the same destination pulls along on a parallel channel. After the block is cleared, the message arriving later may go first because of the round-robin link allocation policy, which does not guarantee FCFS service.

Out-of-order arrival of messages causes problems if the coherence protocol is not modified. For example, a situation may arise where an out-of-order invalidation message reaches a node before the data due to a read request arrived. If the invalidation is acknowledged, it may lead to inconsistent state where a node thinks it has an exclusive access to a block, whereas another node has a copy of the block. We have modified the coherence protocol where the cache controllers detect if a message has arrived out-of-order and holds it to be serviced later. The scheme is similar to the scheme used in MIT Alewife system [14].

The synchronization method used in our simulations is based on spin-locks using test-

test-and-set operation with exponential backoff [11]. Barriers used in many of the applications were implemented using a shared counter. We also experimented with other types of barriers to reduce contention, however, our experience suggests that the main overhead in synchronization is the contention for the lock itself, not for the shared variable in the barrier. The overhead due to contention on synchronization variables is significant for some applications, such as MP3D. The network latency also plays an important role on the synchronization overhead. We will point out the impact of network latency on the synchronization overhead while discussing our simulation results.

3 The Benchmark Applications

We have selected some numerical applications as the workload for evaluating the network performance in a cache-coherent shared-memory environment. These applications are multiplication of two 2-D matrix (MATMUL), Floyd-Warshall's all-pair-shortest-path algorithm (FWA), blocked LU factorization of a dense 2-D matrix (LU), 1-D fast Fourier transform (FFT), and simulation of rarefied flows over objects in a wind tunnel (MP3D).

The matrix multiplication was done between two 128×128 double precision matrices. The principal data structures are four shared two-dimensional arrays of real numbers: two input matrices, a transpose matrix, and one output matrix. The shared data size is about 512 Kbytes. The problem was partitioned into square blocks of the output matrix and all the elements in a block are computed by one processor. There is no real sharing of writable data, but there may be some false sharing of cache blocks at the boundaries of partitions. This type of partition minimizes the amount of shared data accessed by each processor. One of the input matrix is transposed to avoid large number of conflict misses.

For Floyd-Warshall's algorithm, we used a graph of 128 nodes with random weights assigned to the edges. The principal data structures are two shared two-dimensional arrays of integers: one distance matrix and another predecessor matrix. The shared data size is about 128 Kbytes. The distance matrix is initialized with the weights of the edges in the graph. The problem is partitioned as per the rows in the distance matrix, so a set of vertices is statically scheduled to a processor to compute the shortest paths from them. Therefore, the elements in the distance and predecessor matrices are modified by only one processor during the whole execution. The program goes through as many iterations as the number of vertices, and during an iteration a particular row of distance and predecessor matrix is read by all the processors. Each iteration is followed by a barrier.

The blocked LU decomposition application from SPLASH-2 suite [15] was done on a 256×256 matrix using 8×8 blocks. The principal data structure is a two-dimensional array in which the first dimension is the block, and the second contains all data points in that block. In this manner, all data points in a block (which are operated on by the same

Application	Memory references	Cache misses	Messages generated
MATMUL	9,215,571	743,258	1,474,220
FWA	11,050,133	323,159	739,624
FFT	2,146,864	394,972	1,093,376
MP3D	39,817,997	1,993,443	4,157,153
LU	112,270,761	706,579	1,671,854

Table 1: Characteristics of the applications used in the evaluation.

processor) are allocated contiguously, and false sharing and line interference is eliminated.

We implemented Cooley-Tukey 1-D FFT algorithm. The simulations are done on an input of 2^{14} points. The principal data structures are two 1-D array of complex numbers. There is no data sharing during first $\log_2(N - P)$ stages, where N is the number of data points and P is the number of processors. In rest of the $\log_2 P$ stages, every data point is shared by two processors. During these stages, instead of using two separate input and output array, we interleave these arrays to avoid large number of conflict misses. Though, this algorithm is not optimal for cache based systems [16], but for given problem size, number of processors and cache size, it performs fairly.

MP3D [17] is a three-dimensional particle simulator used in rarefied fluid flow simulation. We used 16000 molecules with the default geometry provided with SPLASH [17] which uses a $14 \times 24 \times 4$ (2646-cell) space containing a single flat sheet placed at an angle to the free stream. The simulation was done for 5 time steps. In this application, the active space is divided into cells and the molecules interact only with the molecules in the same cell. There are two principal data structures: one stores the state information of each molecule, and the other stores the properties of each cell. The shared data set size is about 800 Kbytes. The work is partitioned by molecules, which are statically scheduled on processors. A clump size of 8 was used. The LOCKING option was not used to avoid contention for the locks allowing some race conditions that can be overlooked in the simulation.

Some of the relevant characteristics of these applications are shown in Table 1. It shows the number of shared memory references, the cache misses on shared memory references, and the total number of messages generated during the execution. We would like to point out here that the number of messages differs for different network configurations due to changes in synchronization and dynamic nature of coherence protocol with busy messages and retries. However, these variations are usually small. The numbers presented here is for the wormhole routed network with 1 flit buffer per link without any virtual channels.

Fig 2 shows the traffic pattern between every pair of nodes for MATMUL. The x-axis is the destination node number, the y-axis is the source node number, and the z-axis represents

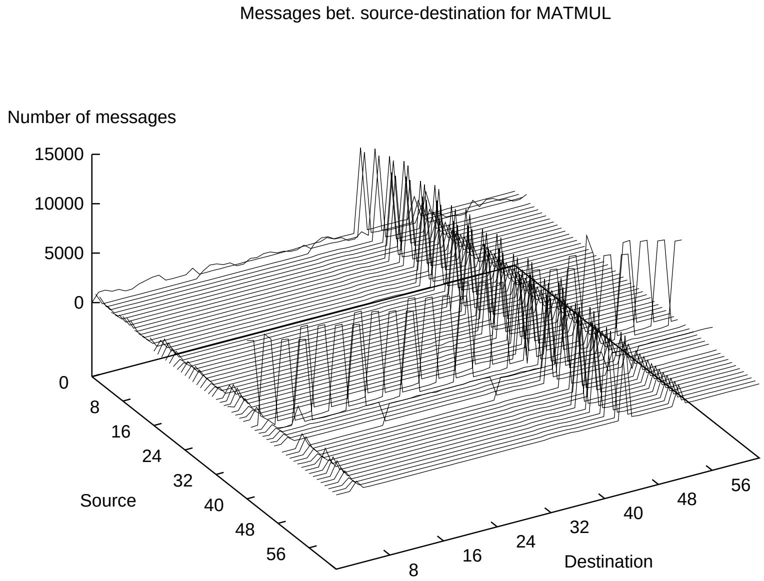


Figure 2: Traffic pattern for MATMUL.

the number of messages between a source-destination pair. All the memory blocks used by the application are located on few of the nodes, resulting in the concentration of messages to and from those nodes.

The traffic pattern for FWA is shown in Fig 3. All the traffic is concentrated to and from only few of the nodes. Apart from cold misses, most of the messages in this application is generated due to invalidations, acknowledgments, and read requests following a write.

The traffic pattern for FFT is shown in Fig 4. The traffic is concentrated to and from a small number of nodes. Eventhough the cache hit rate is not very good, Table 1, the cache misses are generated at regular intervals and most of the writes generate only a small number of invalidations. The sharing of block is usually between just 2 processors. Therefore, there are fewer hot-spots in the network.

Figure 5 shows the traffic pattern for the MP3D. The traffic is concentrated to and from few of the nodes. The noticeable feature is the large number of messages to and from node 0 to all the other nodes. This is due to contention over the synchronization semaphore that is located at node 0. We used test-test-and-set with exponential backoff to acquire semaphore.

The traffic pattern for LU is shown in Fig 6. The memory is allocated in terms of small blocks in this application. The memory allocation scheme maintains a list of buckets of free memory and a request is satisfied by first-fit approach. This distributes the blocks to different nodes in the system. Therefore, the traffic is distributed over several nodes, not to just a few nodes as in other cases. However, each node communicates with only a small

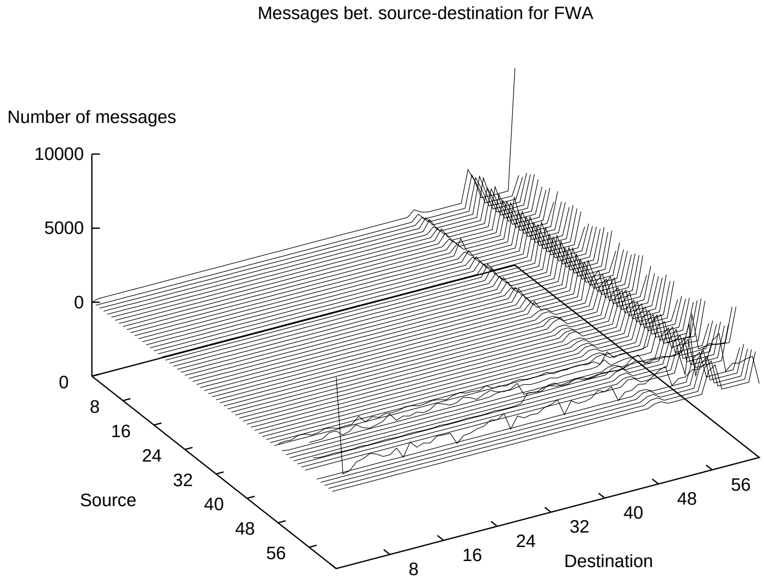


Figure 3: Traffic pattern for FWA.

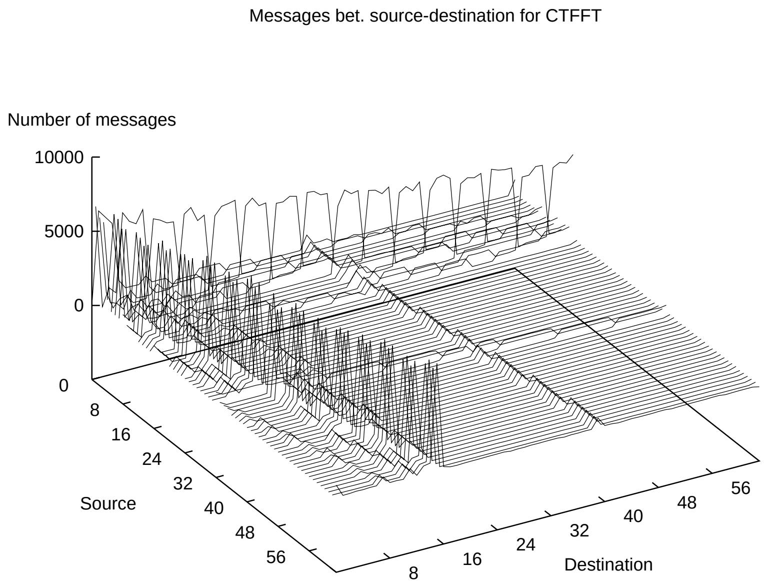


Figure 4: Traffic pattern for FFT.

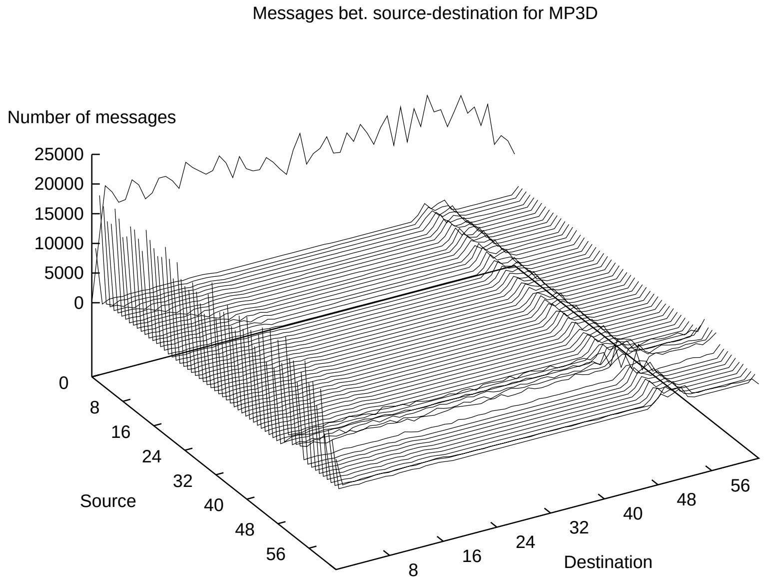


Figure 5: Traffic pattern for MP3D.

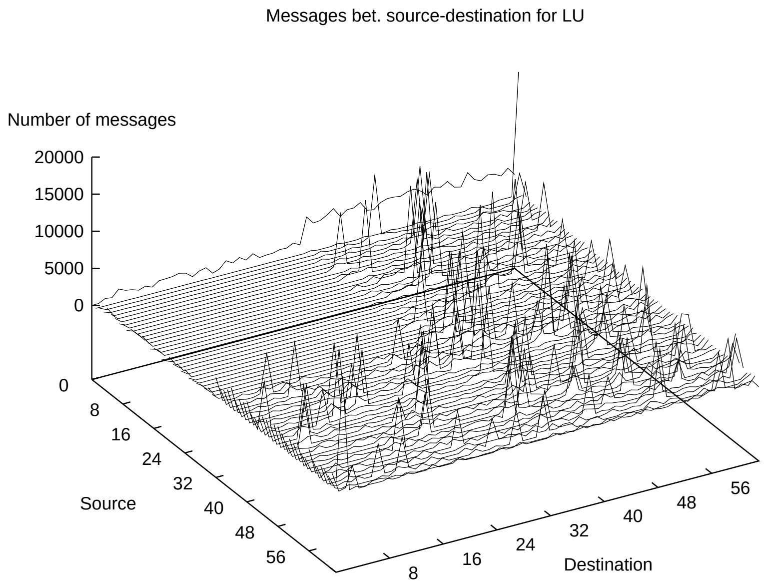


Figure 6: Traffic pattern for LU.

number of nodes. Each node communicates with only a few other nodes. The traffic from and to node 0 is much higher due to the location of synchronization semaphore at node 0.

4 Measurement of Message Latencies

All the data presented in this paper takes into account only the parallel sections of these applications, including the synchronization overhead. The parallel applications were executed for different network parameters. In case of wormhole routing, the measurements were done by varying the number of virtual channels and flit buffers per virtual channel. For packet switching, the buffer size per link was varied. In this section we present the average message latencies for various applications on different network architectures. A message can be a control message or a data message, as explained in section 3. The latency is the time from the generation of a message until the message is received at the destination. The simulation parameters used are presented below.

4.1 Simulation parameters

The system parameters used in the simulation are listed in Table 2. We simulated a 64 node system using a 6 stage MIN with 8KB of cache and 1MB of memory per node. A smaller cache size was selected since the applications used smaller data sets to keep the simulation time manageable. The cache line size of 32 bytes and a 2-way set associative organization was used. A switching delay of 1 cycle was assumed to make the routing decision for the first flit of a message. The subsequent flits do not see the switching delay. A flit size of 16 bits was considered. The links were also 16 bits wide, so transferring a flit on the link took 1 cycle. In a cache-coherent system, the messages are of two different lengths. The data messages containing the memory block is longer, while the control messages, with only the address and protocol information, is shorter. We assumed message lengths of 8 and 40 bytes for control and data messages, respectively. For packet switching, the packets were 8 bytes long, so a packet sees a delay of (1+4) cycles for switching and transmission to the next stage. Also, we assumed that each packet has 1 byte of header and rest 7 bytes are used by data.

In case of wormhole routing, the number of virtual channels were varied from 1, 2, 4, 8, and 16. The number of flits buffers per virtual channel were from 1, 2, 4, 8, and 16. For packet switching, the buffer size per link were 8, 16, and 32 bytes.

4.2 Performance of packet switching and wormhole routing

Table 3 shows the message latencies for packet switched and wormhole routed networks. The performance of packet switched network is evaluated with variable number of packet

Parameter	Value
Number of processors	64
Shared memory size per node	1 Mbytes
Cache size	8 Kbytes
Cache line size	32 bytes
Set size	2
Cache access time	1
Memory access time	8
Switching delay	1
Link width	16 bits
Flit length	16 bits
Packet size	8 bytes
Message lengths	8 or 40 bytes

Table 2: Simulation parameters.

buffers per link. The measurements show that the average message latency decreases with increase in the buffer capacity for all applications. The largest decrease in the latency is observed when the number of packet buffers per link is increased from 1 to 2. The reason for this large change is that when the buffer size is more than 1, a switch can be receiving a packet at the same time as it is sending a packet to the next stage. If there is only one buffer, then a switch must wait for the packet to move to the next stage before receiving another packet. The message latencies keep decreasing with further increase in the buffering capacity, but the gain from 2 buffers to 4 buffers is much smaller.

Another noticeable characteristics is the large variation in the message latencies across the applications. This is because of the change in traffic pattern due to the sharing and coherence characteristics of these applications. Some of the applications, such as FWA and MATMUL, generate very bursty traffic and create hot-spots in the network, resulting in much higher message latencies compared to other applications.

Table 3 also shows the average message latency for wormhole-routed network, with different number of flit buffers per channel. Increasing the number of flit buffers per channel generally decreases the average message latency. The decrease is prominent when the number of flit buffers are increased from 1 to 2, as was observed in the case of packet switching. The gains are smaller as the number of flit buffers are increased further. The improvement is the result of the shorter tails on the blocked messages, so fewer channels are occupied by a blocked message. This makes more channels available for movement of flits. Increasing the number of flit buffers beyond 4 results in very little gain. This is because

Packet switching					
Packet buffers	FFT	FWA	LU	MATMUL	MP3D
1	599.6	3437.1	329.1	2022.6	794.5
2	275.7	1463.2	186.8	574.5	371.6
4	266.3	1433.9	185.9	555.1	352.6
Wormhole routing					
Flit buffers	FFT	FWA	LU	MATMUL	MP3D
1	136.6	539.5	71.0	226.6	308.3
2	69.6	286.3	40.9	80.7	175.7
4	58.9	220.4	37.1	55.7	149.5
8	57.7	236.0	36.6	53.8	147.7
16	57.2	214.7	35.8	51.9	148.8

Table 3: Average message latencies for packet-switched and wormhole-routed networks.

majority of the messages (control messages) are 4 flits long, so they can be stored at one switch without blocking other links. Again, the message latencies show a large variation across the applications due to their differing sharing characteristics.

The comparison of average message latencies for packet switching and wormhole routing shows that wormhole routing outperforms packet switching even with much smaller storage per switch. Even 4 packet buffers (32 bytes) per link for packet switching performs worse than 1 flit buffer (2 bytes) per link for all the applications. The reason is that packet switching has the overhead of putting header (1 byte) with every packet which increases the amount of data transferred. Also, since each packet is routed individually, every packet has a switching delay of 1 cycle. In case of wormhole routing, a message sees only one switching delay at any switch. Also, for packet switching, large messages that are divided into multiple packets, the buffers and link bandwidth gets shared by multiple messages in the flow at the same time. This may increase the average message latency if there are hot-spots in the network and the traffic is bursty. In case of wormhole routing, the allocation of link bandwidth to messages are much more controlled and fewer messages could be in flow at any time.

Therefore, our measurements clearly indicates that wormhole routing performs better than packet switching with smaller average message latency. Wormhole routing results in better link and storage utilization, thus should be cheaper to build, resulting in superior cost performance ratio.

Virtual channels	Flit buffers	FFT	FWA	LU	MATMUL	MP3D
1	1	136.6	539.5	71.0	226.6	308.3
2	1	61.2	231.7	45.3	57.0	142.0
4	1	51.1	173.2	43.9	53.2	98.7
8	1	52.6	167.6	45.2	53.8	96.5
16	1	53.2	157.2	44.7	54.8	99.9
4	1	51.1	173.2	43.9	53.2	98.7
4	2	46.3	175.8	33.2	43.3	97.2
4	4	44.5	169.6	32.3	41.1	97.0
4	8	44.7	173.0	33.4	40.5	97.0

Table 4: Average message latencies for wormhole-routed network with virtual channels.

4.3 Performance of virtual channels

We have also evaluated the performance of wormhole networks with virtual channels. Table 4 shows the average message latencies for each of the applications for different number of virtual channels and flit buffers. The first set of measurements were done for different number of virtual channels, keeping the number of flit buffers to 1. Increasing the number of virtual channels usually decreases the average message latency. Here, the improvement is achieved by providing alternate buffers to messages and allowing them to bypass a blocked message. There is significant improvement when the number of channels are increased from 1 to 2, but the improvement is smaller for 4, 8 and 16 virtual channels. In fact, in some cases the average message latency even increases for larger number of virtual channels. Several factors are responsible for this unusual behavior.

One of the reason is the segmentation of worms which results in poor buffer utilization. As a message gets blocked in the network, it holds the buffer resources, but releases the link to be used by other channels. If the blocked message is spanned over multiple routers, all the links may not be immediately available to the corresponding channels when the block is released. The link in the path may be allocated to the corresponding channels at different times, segmenting the worm and creating bubbles of idle buffers in the stream. These idle buffers cannot be used by any other message and wastes the buffer resources. Also, because of the round-robin link allocation among multiple virtual channels, the average latency may increase due to sharing of link bandwidth among larger number of messages as the number of virtual channels are increased.

In another set of experiments, we fixed the number of virtual channels to 4 and varied the flit buffer size. Increase in the size of flit buffers per channel generally reduces the average message latency. This is because of the shorter tails on the blocked messages and fewer segmentation of worms in the presence of virtual channels. The improvement is appreciable up to 4 flit buffers since the control messages are 4 flits long, allowing complete control messages to be stored at one switch. In some cases increasing the buffers per channel beyond 4 leads to increase in message latency. The reason for this phenomena is very much dependent on the application characteristics. As per these measurements, 4 virtual channels and 4 flit buffers per channel seems to be the best configuration for most of the applications.

When the size of flit buffer per channel is increased beyond 4, it increases the message latency in some applications. In case of congestion, the worms get segmented as explained earlier. Now, the links are assigned to these smaller messages of the size equal to or less than the buffer size. It should be noted here that all the segmented worms are not of the same size. Majority of the messages are just 4 flits long in cache-coherent systems. Therefore, the queue at the links contains jobs of different size, some of size 4 and others of size equal to the buffer size. In this situation, the best scheduling policy is the shortest-job-first scheme, and the round-robin link allocation scheme is not optimal. The performance of round-robin scheme deteriorates as the difference in size of jobs become larger, which suggests that increase in the buffer size per channel can deteriorate performance. The smaller messages get behind larger segments of large messages in acquiring the link, and see increased waiting times.

5 Measurement of Execution Times

In this section, we will discuss the impact of switching techniques and network parameters on the execution time of various applications. The previous section presented the results in terms of the average message latency. As one can notice in this section, the relative impact of network architecture on execution time is quite different than on the message latency. This justifies our approach of evaluating multiprocessor networks in an application driven environment in terms of their execution time, rather than based on message latency and network throughput. The main reason for the difference between the characteristics of execution time and message latency with respect to network architecture is that the execution time of an application does not depend on the individual messages but on a collection of messages. For example, in a cache coherent system, the time to complete a write operation on shared data does not depend on the time of individual invalidations but it depends on the time to complete all the invalidations. Therefore, even if a packet switched network may exhibit much higher message latency compared to a wormhole routed network, the execution time of an application using these network could be comparable.

The overall execution time of MATMUL for different network architectures is shown in

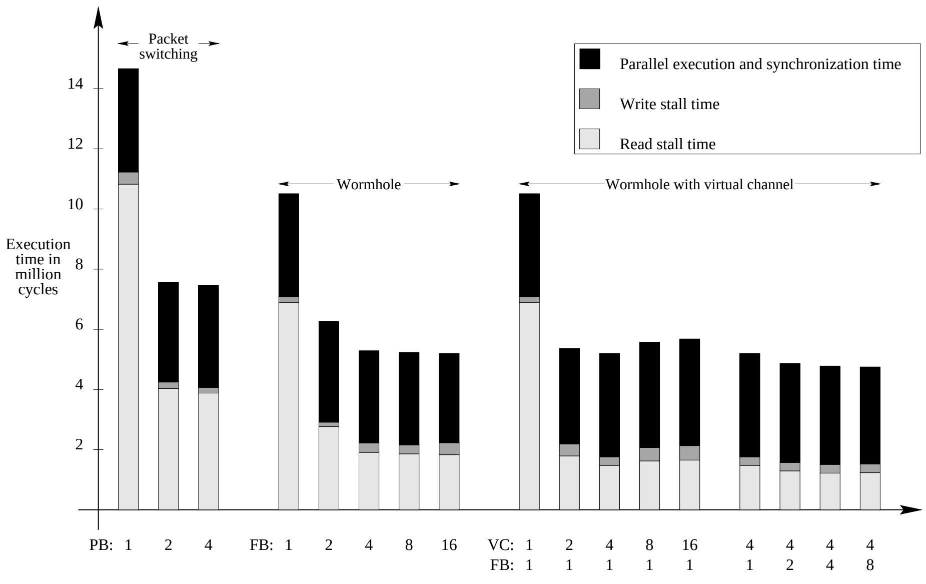


Figure 7: Execution time of MATMUL for different network architectures.

Fig 7. It shows the time spent in computation and synchronization, the read stall time, and the write stall time, separately. The read and write stall times shown here are the maximum read and write stall times observed by any processor. The first set of bars show the execution times for packet switched network for variable number of packet buffers per link. As the number of packet buffer is increased from 1 to 2, the execution time decreases considerably because of the reduction in read and write stall times. However, the change in execution time from 2 to 4 packet buffers is very small. The second set of bars show the execution time for wormhole routed network for variable number of flit buffers. The execution time decreases with increase in the number of flit buffers. The improvement is noticeable up to 4 flit buffers, however, beyond 4 flit buffers the gain is very small. Flit buffer size of 4 is enough to store majority of the messages (control messages) in a switch, eliminating most of the blocking from the network. The third set of bars show the performance of wormhole network with virtual channels. The execution time decreases considerably when the number of virtual channels are increased from 1 to 2, keeping the number of flit buffers per channel to 1. Further increase in the number of virtual channels to 4 causes only a small change and for the values of 8 and 16 the execution time is in fact larger. This is because of the segmentation of worms discussed in the previous section. When the number of virtual channels is fixed to 4, the increase in flit buffers per channel leads to decrease in the execution time. Again the change in execution time is small beyond 4 flit buffers per channel.

The execution time of FWA for different network architectures is shown in Fig 8. For

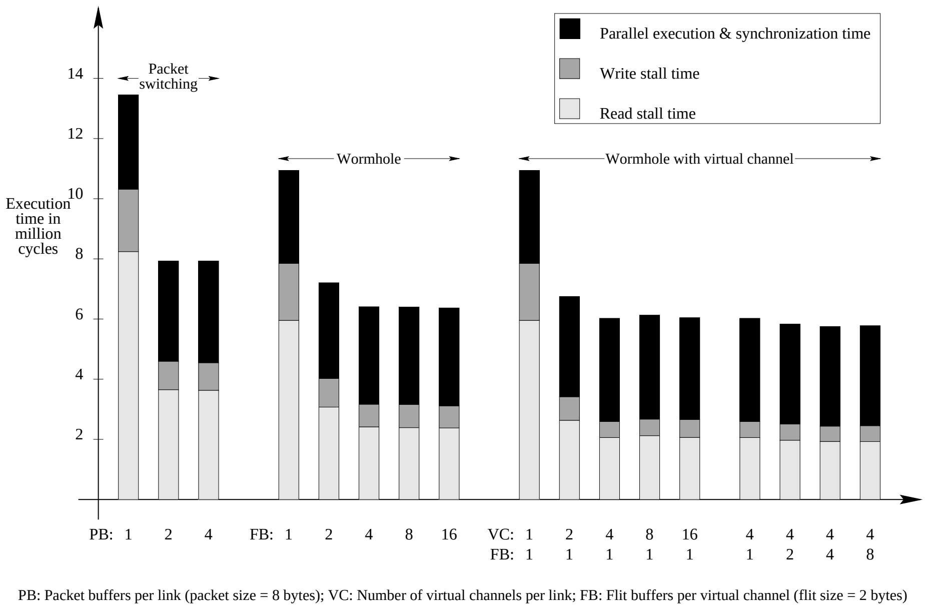


Figure 8: Execution time of FWA for different network architectures.

packet switched network, as the number of packet buffer is increased from 1 to 2, the execution time decreases considerably because of the reduction in read and write stall times. However, the change in execution time from 2 to 4 packet buffers is very small. For wormhole routed network, the execution time decreases with increase in the number of flit buffers. The improvement is noticeable up to 4 flit buffers, however, beyond 4 flit buffers the gain is very small. For wormhole network with virtual channels, the execution time decreases with increase in number of virtual channels up to 4, keeping the number of flit buffers per channel to 1. Further increase in the number of virtual channels does not have any impact on the execution time. When the number of virtual channels is fixed to 4, the increase in flit buffers per channel leads to decrease in the execution time. Again the change in execution time is noticeable only up to 4 flit buffers per channel.

The execution time of FFT for different network architectures is shown in Fig 9. For packet switched network, as the number of packet buffer is increased from 1 to 2, the execution time decreases considerably. However, the change in execution time from 2 to 4 packet buffers is very small. For wormhole routed network, the execution time decreases with increase in the number of flit buffers. The improvement is noticeable up to 4 flit buffers, however, beyond 4 flit buffers the gain is very small. For wormhole network with virtual channels, the execution time decreases with increase in number of virtual channels up to 4, keeping the number of flit buffers per channel to 1. Further increase in the number of virtual channels does not have any impact on the execution time. When the number of virtual channels is fixed to 4, the increase in flit buffers per channel leads to decrease in the execution time. Again the change in execution time is noticeable only up to 4 flit buffers

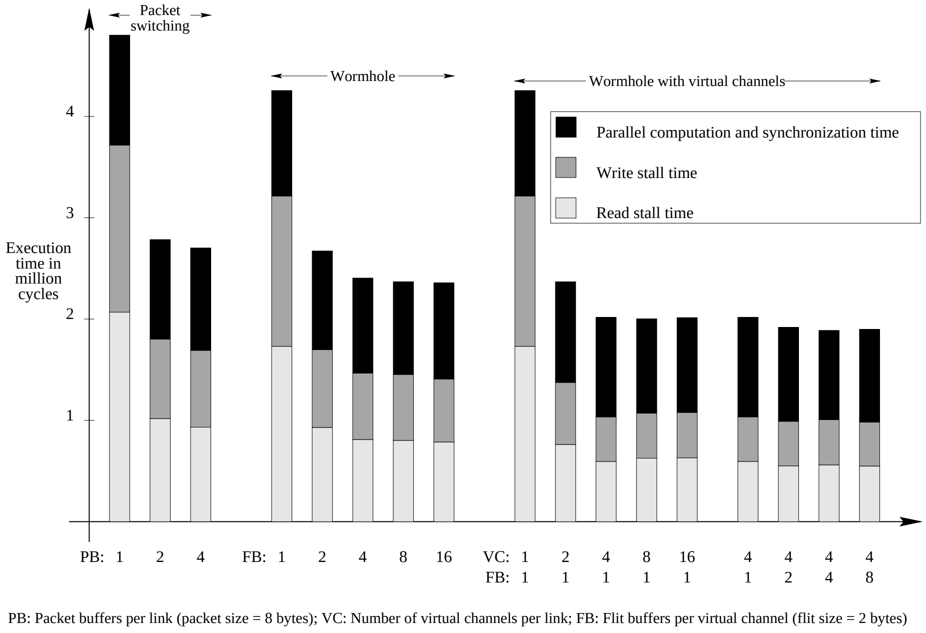


Figure 9: Execution time of FFT for different network architectures.

per channel.

The execution time of MP3D for different network architectures is shown in Fig 10. For packet switched network, as the number of packet buffer is increased from 1 to 2, the execution time decreases considerably. Further increase in the number of packet buffers results in diminishing returns. For wormhole routed network, the execution time decreases with increase in the number of flit buffers. The improvement is noticeable up to 4 flit buffers, however, beyond 4 flit buffers the gain is very small. For wormhole network with virtual channels, increasing the number of virtual channels up to 4 decreases the execution time, however, the execution time for 8 and 16 virtual channels are more than the execution time for 4 virtual channels. The reasons for this unusual response is the segmentation of worms and increase in the number of messages, as explained in the previous section. Also, virtual channels cause messages to arrive out-of-order from a source to a destination. This can cause some of the memory requests to arrive earlier than the directory could be updated due to an invalidation or replacement. The coherence protocol [14] responds with a busy message and that request must be retried later. Such retrials causes increase in the number of messages and deteriorates the congestion at the node containing the synchronization semaphore, which is already bad in case of MP3D. When the number of virtual channels is fixed to 4, the increase in flit buffers per channel leads to decrease in the execution time. Again the change in execution time beyond 4 flit buffers is very small.

A faster network also results in faster synchronization. This can be seen from the lower computation and synchronization time in Fig 10 for the network configurations that have

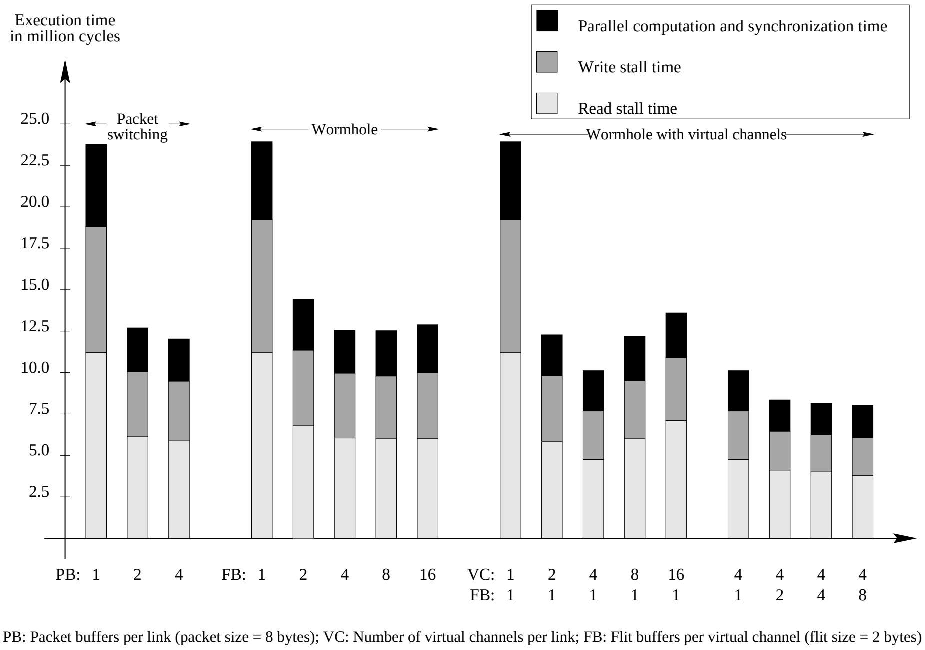


Figure 10: Execution time of MP3D for different network architectures.

smaller read and write stall times. Since the computation time does not change with the change in the network, this change can only come from the synchronization time.

The execution times of LU for different network architectures is shown in Fig 11. In contrast to other applications, the impact of network architecture on the execution time is much smaller for LU. The cache hit ratio is extremely good in this case, resulting in very few read or write stalls. For packet switched network, as the number of packet buffer is increased from 1 to 2, the execution time decreases considerably. However, the change in execution time from 2 to 4 packet buffers is very small. For wormhole routed network, the execution time decreases with increase in the number of flit buffers. The improvement is noticeable up to 2 flit buffers, beyond which the gain is very small. For wormhole network with virtual channels, the execution time decreases with increase in number of virtual channels up to 2. Further increase in the number of virtual channels does not have any impact on the execution time. When the number of virtual channels is fixed to 4, the increase in flit buffers per channel leads to decrease in the execution time. Again the change in execution time is noticeable only up to 2 flit buffers per channel.

6 Conclusions

In this paper we evaluated the performance of a multistage interconnection network using execution-driven simulation in a cache-coherent shared-memory environment for some

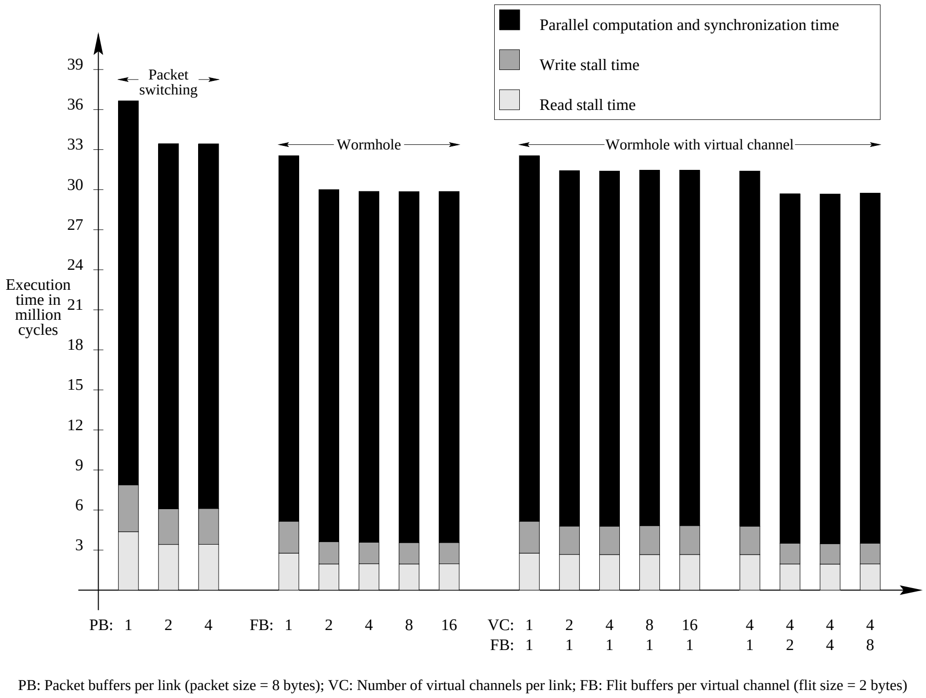


Figure 11: Execution time of LU for different network architectures.

scientific applications. One of the important observation in our study is that wormhole routing results in much better performance compared to packet switching for comparable amount of buffer space per switch. Wormhole routing shows much better average message latency compared to packet switching. In terms of the execution time of the applications, the wormhole routing is still better than the packet switching but not by the same margin as observed for message latencies. In fact some of the configurations of packet-switched network performs better than the configurations of wormhole-routed network in terms of the execution time, even though, the average message latency of the packet-switched network is never better than any configuration of wormhole network. It is because the execution time of an application depends on the time to deliver a collection of related messages, not just individual messages. This observation supports the motivation behind this paper for an execution-based evaluation of the network.

It is noticed that both the message latency and execution time are reduced by increasing the buffer length in the MIN switches. The maximum effect is felt when the buffer size is increased from 1 to 2 units for both packet-switched and wormhole networks. Further increase in buffer space has very small effect on the performance, which is consistent with the results reported in [7, 6].

Another important observation from our study is that the virtual channels in wormhole networks does help in reducing the execution time in shared-memory multiprocessors. About 4 virtual channels seems to offer the best performance in most of the cases. Further increase in the number of virtual channel does not result in appreciable performance improvement and in some cases it even deteriorates the performance. This is because of the segmentation of worms at high load which results in poor buffer utilization. Increasing the number of flit buffers per virtual channel is very effective in reducing the average message latency and the execution time. It is observed that 2 to 4 flit buffers per virtual channels are usually enough. Further increase in the number of flit buffers has only a small impact on the performance, and in some cases it may even degrade the performance. Also, given a constant amount of buffer space, it is necessary to properly balance the number of virtual channels and flit buffers per channel to obtain the best performance.

The execution based evaluation of the network in this paper shows the impact of various network architectures and also points the ways to further improve the performance. Our measurements show that the utilization of the network is very low for most of the applications. Even at this low utilization, the waiting time is sometimes very high due to bursty nature of the traffic in shared-memory systems. We are considering various message scheduling policies and memory interleaving schemes to assess their impact on the execution of applications.

References

[1] D. J. Lilja, "Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons," *ACM Computing Surveys*, vol. 25, no. 3, pp. 303–338, September 1993.

[2] L. N. Bhuyan, Q. Yang, and D. P. Agrawal, "Performance of Multiprocessor Interconnection Networks," *Computer*, pp. 25–37, February 1989.

[3] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb. 1993.

[4] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[5] A. K. Nanda and L. N. Bhuyan, "Design and Analysis of Cache Coherent Multistage Interconnection Networks," *IEEE Transactions on Computers*, pp. 458–470, April 1993.

[6] J. Ding and L. N. Bhuyan, "Finite Buffers Analysis of Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. 43, no. 2, pp. 243–247, February 1994.

[7] Y. Mun and H. Y. Youn, "Performance Analysis of Finite Buffered Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. 43, no. 2, pp. 153–162, February 1994.

[8] V. S. Adve and M. K. Vernon, "Performance Analysis of Mesh Interconnection Networks with Deterministic Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 3, pp. 225–246, March 1994.

[9] J. Torrellas and Z. Zhang, "The Performance of the Cedar Multistage Switching Network," In *Proceedings Supercomputing '94*, pp. 265–274, Washington, DC, November 1994, IEEE Computer Society.

[10] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, vol. C-27, no. 12, pp. 1112–1118, December 1978.

[11] J. M. Mellor-Crummey and M. L. Scott, "Synchronization Without Contention," In *Proceedings of Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 269–278, Santa Clara, CA, April 1991, ACM.

[12] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl, "PROTEUS: A High-Performance Parallel-Architecture Simulator," Technical Report MIT/LCS/TR-516, Massachusetts Institute of Technology, September 1991.

[13] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, October 1991.

[14] J. D. Kubiatowicz, "Closing the Window of Vulnerability in Multiphase Transactions: The Alewife Transaction Store," Technical Report MIT/LCS TR 594, Massachusetts Institute of Technology, February 1993.

[15] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodical Considerations," In *Proc. of the 22nd Annual International Symposium on Computer Architecture*, pp. 24–36, Santa Margherita Ligure, Italy, June 1995, ACM and IEEE Computer Society.

[16] A. Kumar and L. N. Bhuyan, "Parallel FFT Algorithms for Cache Based Shared Memory Multiprocessors," In *Proc. of 1993 Int'l Conference on Parallel Processing*, volume III, pp. 23–27, St. Charles, IL, August 1993, CRC Press, Inc.

[17] J. P. Singh, W.-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 1, , March 1992.