# Effect of Virtual Channels and Memory Organization on Cache-Coherent Shared-Memory Multiprocessors *

Akhilesh Kumar and Laxmi N. Bhuyan

**Technical Report # 96-005**

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA.
E-mail: {akhil,bhuyan}@cs.tamu.edu

February 20, 1996

**Abstract**

In this paper, performance of wormhole routed 2-D torus network with virtual channels has been evaluated for cache-coherent shared-memory multiprocessors with execution-driven simulation using various applications. The traffic in such 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. We show the impact of various network parameters, such as number of virtual channels, number of flit buffers per virtual channel, and number of internal links. We have also considered low-order and high-order interleaving of memory blocks on nodes to show its impact on the network performance. The study shows that 4 virtual channels per link is most efficient for 2-D torus networks. The number of flit buffers per virtual channel also has a considerable impact and 2 to 4 flit buffers are usually enough. The number of internal links also has an impact on the performance for applications, such as MP3D, that generate large contention for shared variables. Larger number of internal links are also useful in case of high-order interleaved memory to reduce hot-spots at the communication interface of favorite nodes.

*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.

Wormhole routing [3] is an efficient technique to reduce the latency of internode messages. In wormhole routing, the messages/packets 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 nodes. Virtual channels [4] are used in wormhole networks to avoid deadlocks and to improve link utilization and network throughput. In this paper we evaluate the performance of a 2-D torus network with wormhole routing and virtual channel flow control in shared-memory multiprocessors. We selected a 2-D torus network with bidirectional links for our performance study, because it is a popular topology [5, 6, 7, 8]. Also, mesh networks without end-around connections have significant performance degradations at the boundary nodes, even under uniform communication [6, 9].

The performance of wormhole networks with virtual channels has been evaluated in various studies [4, 7, 10] in a message-passing environment. However, all these evaluations are based either on analytical models that assume certain traffic distributions or on simulations using simple probabilistic workloads. Adve and Vernon [7] analyzed the performance of mesh and torus networks using closed queueing models. Their model can take into account a limited number of multiple requests from a node before it is blocked. This is closer to the reality both for message-passing and shared-memory environments. However, the model is not appropriate for cache-coherent systems, where a number of invalidation messages are generated to maintain coherence among the caches and the main memory. On the other hand, analytical and simulation models, such as [11, 12], capture the cache coherence traffic in detail, but here we concentrate on wormhole routing with virtual channels. Also, we wish to study the execution of real applications in this paper.

In a related work [13], 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 say how a cache coherence protocol affects the performance of an IN.

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 wormhole routing and virtual channels with proper blocking of 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.

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 only three network parameters – number of virtual channels, number of flit buffers per channel, and number of links between the compute node and the router, called *internal links*. We will report the effect of high-order and low-order memory interleaving policies on the execution time of the applications. The characterization of traffic in cache-coherent systems and the effect of adaptive routing, 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 results of our simulations. Section 5 provides the details about the execution of each application. Finally, section 6 presents the conclusion and the direction for further work in this area.

## 2 Simulator Development

We have modified the Proteus simulator [14] extensively to incorporate virtual channels, multi-flit buffers, multiple internal links and other architectural features. The system considered for evaluation in this paper is a cache-coherent shared-memory multiprocessor connected through a two-dimensional torus network. The network is wormhole routed with virtual channels. The links are bidirectional with separate connections in each direction. In this section we will describe the model of the network interface, the router, and the cache coherence protocol. In the rest of the paper we refer to physical connections between routers 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 router

The node and router architecture used in our simulation model is shown in Fig 1(a). The processing nodes in the system consist of processor, cache, cache controller (CC), a section of the distributed shared-memory including the memory controller (MC), and a network interface. The nodes are connected to a router through the network interface via internal links. The input and output links of a router connect to other routers to form the torus network structure. The system implements a full-map directory-based cache-coherence scheme [15] 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 memory controller in response to remote requests and coherence actions, which are independent of the state of the processor.

We have considered two different distributions of shared memory blocks among the nodes. In one organization, the memory is low-order interleaved across the processors in terms of blocks and the node where a block is located is obtained using the least significant bits. In another organization, the memory is high-order interleaved, so contigous blocks reside on the same node. The size of a memory block is same as the cache line size.

The network interface provides storage 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 flits, initializing header flit with necessary information, etc. The network interface is connected to the router through *internal links*, as shown in Fig 1(a). There could be multiple input and output internal links. In case of multiple input internal link, there are separate queues to store messages at the interface for each internal link.

The buffers at each input of the router is divided into a set of virtual channels. 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 router. The routing decisions and flit movements are done using the following steps during every clock cycle.

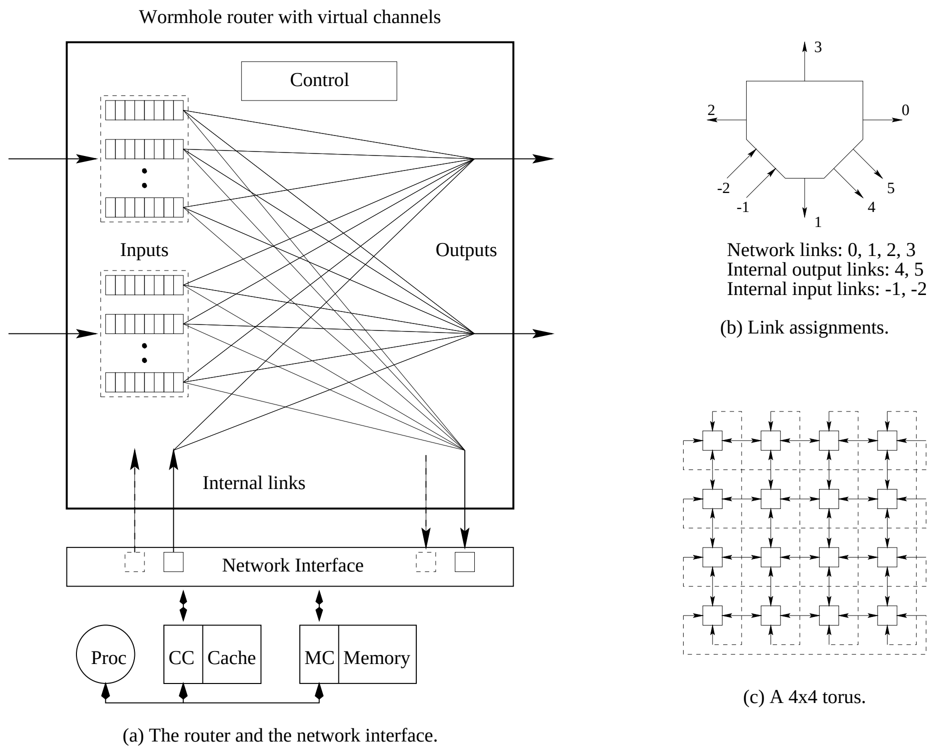



Figure 1: The node architecture and router.

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 flits), 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.

The routing scheme used in this paper is the *e-cube* routing algorithm. The message is always routed in the lowest required dimension, where a required dimension is the dimension in which the coordinates of the current node and the destination node are different. We avoid deadlock in the network by using the scheme proposed by Dally and Seitz [16] where a pair of virtual channels, called low channel and high channel, are used. The same scheme is used when the number of virtual channels is two, otherwise, we divide the virtual channels into odd and even virtual channels. A message using the $i$th dimension link is routed on odd channels if the $i$th digit of the destination node is greater than the $i$th digit of the current node, otherwise, the message is routed on even channels.

## 2.2 Cache coherence protocol and synchronization

We implemented the full-map directory-based cache coherence protocol [15] 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 [17].

The synchronization method used in our simulations is based on spinlocks using test-test-and-set operation with exponential backoff [18]. 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 Workload Environment

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 $128 \times 128$ 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 [19] was done on a $256 \times 256$ matrix using $8 \times 8$ 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 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,

| Application | Memory references | Cache misses | Miss ratio | Data messages | Coherence messages |
|---|---|---|---|---|---|
| MATMUL | 9,194,699 | 722,797 | 7.86% | 748,777 | 727,106 |
| FWA | 11,043,698 | 314,511 | 2.85% | 340,300 | 641,356 |
| FFT | 2,144,606 | 394,625 | 18.40% | 553,602 | 632,260 |
| MP3D | 37,399,880 | 2,024,690 | 5.41% | 1,070,705 | 2,922,032 |
| LU | 111,171,678 | 714,197 | 0.64% | 870,801 | 852,912 |

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

we interleave these arrays to avoid large number of conflict misses. Though, this algorithm is not optimal for cache based systems [20], but for given problem size, number of processors and cache size, it performs fairly.

MP3D [21] is a three-dimensional particle simulator used in rarefied fluid flow simulation. We used 16000 molecules with the default geometry provided with SPLASH [21] 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, the miss ratio, and the number of data (20 flits long) and coherence (4 flits long) 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 number presented here is for the base configuration, i.e., when the number of virtual channels is 2, number of flit buffers per channel is 1 and there is only one internal input and output link.

MATMUL shows a fairly good cache hit rate since one of the input matrices has been transposed to avoid conflict misses due to line interference. FWA also shows a fairly good hit rate for shared data and thus generates few messages. One important characteristics of this application is the large number of invalidations generated on some of the blocks. FFT shows higher miss rates and larger number of messages. Most of the misses are generated due to the changing access pattern on the shared data during successive iterations. MP3D generates a large number of memory references and has a fairly good hit rate. Most of these cache misses are generated due to contention on locks used by the algorithm. LU generates the largest number of memory references and has a very good cache hit rate. The blocking of data eliminates false sharing and cache line interference.

# 4 Results and Discussions

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 values of number of virtual channels, flit buffers per virtual channel, and number of internal links. Apart from measuring the overall execution time for each execution, we also measured the average message latency, the utilization for each network and internal link, the average waiting times at each link, and the time spent in computation and read and write stalls. Tables 3 and 4 show the average message latency and waiting time for different network configurations.

## 4.1 Simulation parameters

The system parameters used in the simulation are listed in Table 2. We simulated an $8 \times 8$ torus network with 8 KB of cache and 32 KB 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 in each direction, 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 coherence messages, with only the address and protocol information, is shorter. We assumed message lengths of 4 and 20 flits for coherence and data messages, respectively.

## 4.2 Effect of virtual channels

Tables 3 and 4 shows the average message latency and waiting time for different configurations. We started with 2 virtual channels that is necessary to avoid deadlocks in a torus network. We have assumed that there is no overhead for switching link between the channels sharing it. The measurements were done for virtual channels 2, 4, 8, and 16, keeping the buffer size and the number of internal links to 1. To analyze the effect of buffer size we kept the number of virtual channels to 4 and internal links to 1. The buffer size used were 1, 2, 4, and 8. The effect of internal links were studied for the values of 1, 2, and 4, keeping the number of virtual channels and buffer sizes to 4.

| Parameter | Value |
|---|---|
| Number of processors | 64 |
| Shared memory size per node | 32 Kbytes |
| 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 |
| Virtual channels per link (VC) | 2, 4, 8, 16 |
| Flit buffers per channel (FB) | 1, 2, 4, 8 |
| Internal links (IL) | 1, 2, 4 |
| Message lengths | 4 or 20 flits |

Table 2: Simulation parameters.

| Virtual | Flit | Internal | MATMUL | | FWA | | FFT | | MP3D | | LU | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| channels | buffers | links | LOI | HOI | LOI | HOI | LOI | HOI | LOI | HOI | LOI | HOI |
| 2 | 1 | 1 | 50.8 | 302 | 135.0 | 538 | 84.9 | 130 | 235.3 | 314 | 73.2 | 72 |
| 4 | 1 | 1 | 44.2 | 141 | 89.8 | 492 | 57.8 | 76 | 212.5 | 276 | 63.1 | 57 |
| 8 | 1 | 1 | 45.0 | 141 | 116.4 | 497 | 56.9 | 74 | 223.9 | 278 | 60.9 | 60 |
| 16 | 1 | 1 | 44.9 | 141 | 88.2 | 484 | 57.0 | 74 | 216.3 | 275 | 61.6 | 61 |
| 4 | 1 | 1 | 44.2 | 141 | 89.8 | 492 | 57.8 | 76 | 212.5 | 276 | 63.1 | 57 |
| 4 | 2 | 1 | 28.7 | 81 | 56.6 | 261 | 39.1 | 50 | 129.7 | 165 | 37.9 | 41 |
| 4 | 4 | 1 | 27.3 | 74 | 43.0 | 207 | 36.0 | 44 | 110.2 | 141 | 34.2 | 37 |
| 4 | 8 | 1 | 29.1 | 74 | 46.0 | 210 | 35.1 | 42 | 112.5 | 138 | 34.6 | 36 |
| 4 | 4 | 1 | 27.3 | 74 | 43.0 | 207 | 36.0 | 44 | 110.2 | 141 | 34.2 | 37 |
| 4 | 4 | 2 | 25.7 | 65 | 32.4 | 71 | 32.7 | 38 | 53.2 | 71 | 28.0 | 28 |
| 4 | 4 | 4 | 25.9 | 63 | 30.6 | 59 | 32.1 | 36 | 43.7 | 59 | 26.8 | 27 |

Table 3: Average message latencies for low-order interleaved (LOI) and high-order interleaved (HOI) memory.

| Virtual channels | Flit buffers | Internal links | MATMUL | | FWA | | FFT | | MP3D | | LU | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | LOI | HOI | LOI | HOI | LOI | HOI | LOI | HOI | LOI | HOI |
| 2 | 1 | 1 | 28.8 | 280.5 | 114.7 | 517.9 | 63.8 | 108.5 | 217.7 | 295.5 | 51.1 | 49.6 |
| 4 | 1 | 1 | 23.7 | 121.9 | 71.0 | 472.0 | 39.4 | 58.6 | 196.2 | 258.8 | 42.0 | 35.4 |
| 8 | 1 | 1 | 24.9 | 121.4 | 98.4 | 477.0 | 39.8 | 58.7 | 208.3 | 261.2 | 40.1 | 39.3 |
| 16 | 1 | 1 | 24.9 | 121.4 | 70.1 | 464.1 | 40.0 | 58.6 | 193.2 | 258.1 | 40.8 | 41.0 |
| 4 | 1 | 1 | 23.7 | 121.9 | 71.0 | 472.0 | 39.4 | 58.6 | 196.2 | 258.8 | 42.0 | 35.4 |
| 4 | 2 | 1 | 10.2 | 70.7 | 40.5 | 243.1 | 24.9 | 38.4 | 115.7 | 150.0 | 18.5 | 21.0 |
| 4 | 4 | 1 | 5.5 | 53.4 | 23.0 | 186.8 | 16.2 | 25.6 | 93.3 | 124.1 | 12.2 | 15.0 |
| 4 | 8 | 1 | 7.2 | 52.9 | 25.8 | 189.3 | 14.4 | 21.7 | 95.3 | 120.4 | 12.5 | 14.5 |
| 4 | 4 | 1 | 5.5 | 53.4 | 23.0 | 186.8 | 16.2 | 25.6 | 93.3 | 124.1 | 12.2 | 15.0 |
| 4 | 4 | 2 | 3.9 | 44.5 | 12.8 | 51.2 | 13.2 | 19.4 | 36.3 | 54.0 | 6.1 | 6.2 |
| 4 | 4 | 4 | 4.3 | 42.5 | 11.1 | 38.8 | 12.7 | 18.7 | 27.4 | 43.0 | 5.0 | 4.7 |

Table 4: Average waiting time per message for low-order interleaved (LOI) and high-order interleaved (HOI) memory.

Increasing the number of virtual channels usually decreases the average message latency and the average waiting time. 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 2 to 4, but the improvement is marginal for 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 increase in the total number of messages due to more memory request retries caused by out-of-order arrival of messages. Another 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.

## 4.3 Effect of buffer size

Increasing the number of flit buffers per virtual channel also improves the performance considerably. 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 number of channels available for movement of flits. Increase in the size of flit buffers per channel also reduces the number of segements in a worm in case of blocking. The improvement is appreciable up to 4 flit buffers since the coherence messages are 4 flits long, allowing complete coherence messages to be stored at one router. Again, 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.

When the size of flit buffer per channel is increased beyond 4, it increases the message latency and waiting times 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 segements of large messages in acquiring the link, and see increased waiting times.

## 4.4 Effect of internal links

The larger number of internal links reduce the average message latency and waiting time only when the traffic patterns consists of many-to-one or one-to-many message patterns, such as large number of invalidations or acknowledgments for a block. This is the reason that increasing the number of internal links makes an appreciable difference in the average latency in case of FWA and MP3D. In case of other applications which do not have such traffic patterns, the larger number of internal links makes only a marginal difference on the message latency. In particular, the technique improves the performance when memory is high-order interleaved since this relaxes the creation of hot-spots at some nodes. The increase in number of internal links cuts down the waiting time by providing alternate paths between the interface and the router. In certain rare cases, such as MATMUL for low-order interleaved memory, increasing the number of internal links from 2 to 4 results in slight increase in the waiting time and message latency. Larger number of internal links injects more traffic in the network and can cause buffer saturation in some regions leading to increased blocking.

## 4.5 Effect of memory interleaving

The distribution of the shared memory blocks and memory allocation policies have a profound impact on the performance of NUMA shared-memory systems. The memory distribution and allocation policy determines the traffic pattern in the network and may create hot-spots in the network and at the memory modules. When the shared-memory blocks are high-order interleaved, most of the shared blocks are located on a small number of nodes which creates a hot-spot in the network and at the memory modules at those nodes. This results in much higher message latencies and waiting times for high-order interleaved memory compared to low-order interleaved memory, as evident from Tables 3 and 4. For some applications, such as MATMUL and FWA, the low-order interleaving of shared-memory results in large reduction in message latencies. For other applications, such as LU, both low-order interleaving and high-order interleaving perform well. Here, the memory allocation policy plays an important role and the shared blocks are distributed among large number nodes in both the cases. The primitive memory allocation policy used by our simulator partitions the whole shared-memory space into buckets of different size and uses a first-fit allocation policy on the list of free buckets. The allocation is oblivious of the identity of the requesting node and the location of the shared block. A sophisticated memory allocation policy based on the affinity of the shared blocks to the nodes using it may have considerable impact on the performance. We are currently investigating several such memory allocation policies.

# 5 Execution Details of Various Applications

## 5.1 MATMUL

The overall execution time of MATMUL for different network parameters is shown in Fig 2 with both low-order and high-order interleaved memory. It shows the time spent in computation and synchronization, the read stall time, and the write stall time, separately. The figure shows one network parameter at a time, keeping other parameters same. First set of bars show the effect of virtual channels for the values of 2, 4, 8, and 16, keeping buffer sizes and number of internal links to 1. The second set shows the effect of buffer size for the values of 1, 2, 4, and 8, keeping number of virtual channels to 4 and internal links to 1. Third set shows the effect of internal links for the values of 1, 2, and 4, keeping number of virtual channels and buffer size to 4. These measurements are repeated for low-order interleaved memory.

Fig 3(a) and 3(b) shows the traffic pattern between every pair of nodes. The x-axis is the destination node number, the y-axis is the source node number, and the z-axis represents the number of messages between a source-destination pair. In case of low-order interleaved memory, the memory blocks are distributed among all the nodes, so the traffic is fairly uniform between nodes. For high-order interleaving, 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 execution time follows similar pattern as seen for the average message latency in

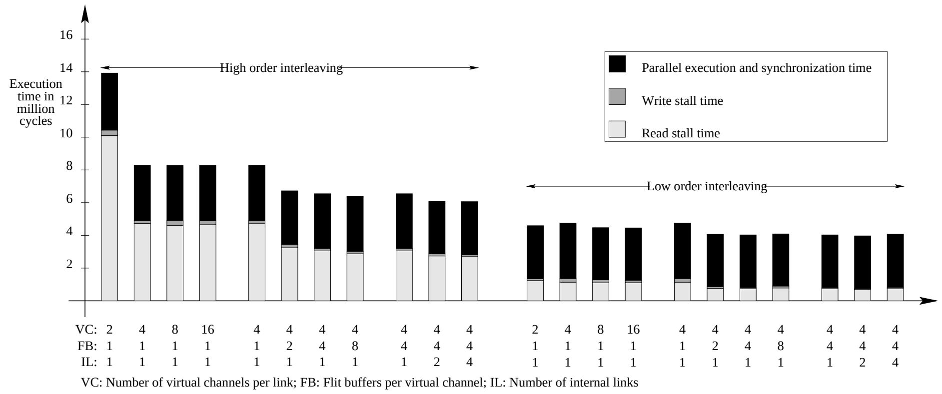
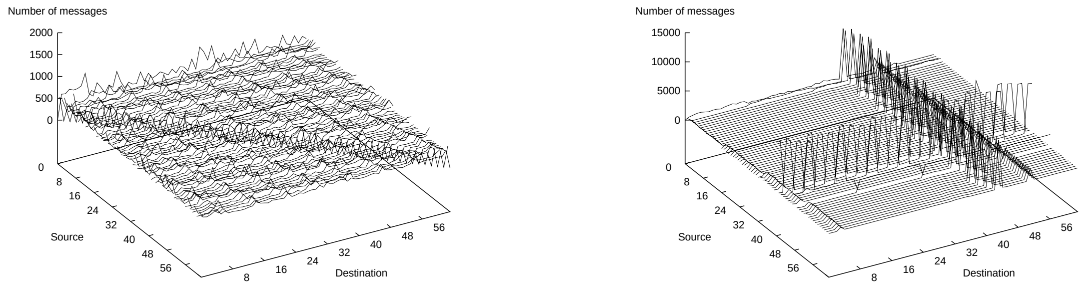



Figure 2: Execution time of MATMUL for various network parameters.




(a) Low-order interleaved memory.

(b) High-order interleaved memory.

Figure 3: Traffic pattern for MATMUL.

Table 3. In case of high-order interleaved memory, the execution time shows a big improvement when the number of virtual channels are increased from 2 to 4. Here, the traffic is concentrated to only few of the nodes that are close to each other. Providing more virtual channels gives alternate paths to bypass a blocked message. In case of low-order interleaved memory, the only network parameter which makes an appreciable difference on the execution time is the number of flit buffers. The traffic is very symmetrical and uniform among the nodes and the traffic intensity is low, so there is very little congestion in the network and almost no blocking of messages. This is supported by the low waiting time per message as shown in Table 4. There is an small increase in the execution time for (4,1,1) over (2,1,1) configuration, eventhough the message latency and waiting times are smaller. This is because of the thrashing on the synchronization variable which causes increase in the number of messages and larger synchronization time.

For this application, most of the time there are very few messages in the network, about

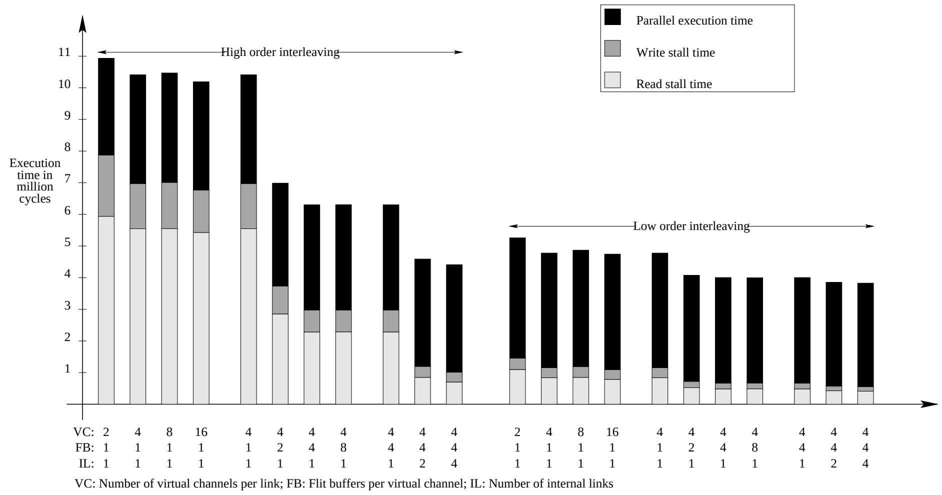



Figure 4: Execution time of FWA for various network parameters.

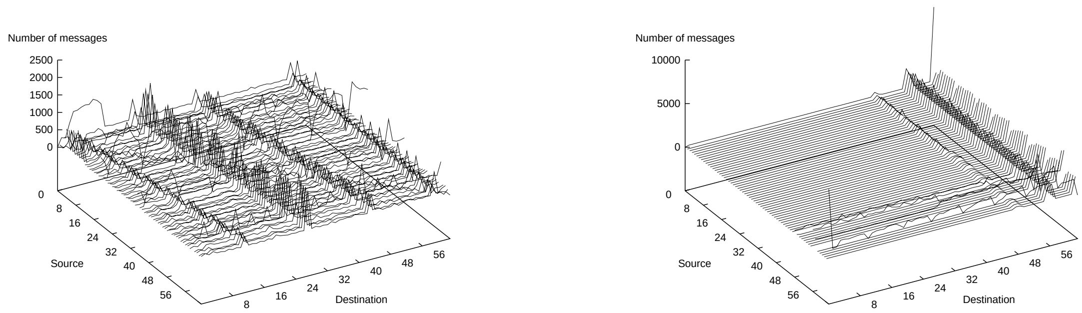
10 to 20, and there are at most 55 to 100 messages at any time in the network.

## 5.2 FWA

The execution time of FWA for different network parameters are shown in Fig 4. Increasing the number of virtual channels from 2 to 4 has only a small reduction in the execution time, whereas in case of MATMUL the reduction was much larger. Increasing the number of virtual channels more than 4 does not have any noticeable benefit. This application has a fairly good hit rate and generates very few messages. However, it generates quite a few write requests on widely shared memory blocks, which generates lot of invalidation messages from memory modules in a burst and later lot of invalidation acknowledgments converge at the memory module. Though the average network utilization is very small, the bursty traffic creates hot-spots in the network. The hot spot is at the interface of memory module and the router, which is the reason for improvement in performance with increase in the buffer size and the number of internal links. As in case of MATMUL, the execution time for FWA reduces considerably by increasing the number of flit buffers per channel from 1 to 2. Again, increasing the number of buffers to 4 or 8 makes only a small difference.

The traffic pattern between every pair of node is shown in Fig 5(a) and 5(b). In case of low-order interleaved memory, the traffic pattern is fairly uniform among nodes except for some nodes. The ridges in the plane are due to shared arrays used for barrier flags and some other data structures used for measuring the computation and synchronization times at each node. In case of high-order interleaved memory, 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.

In this application, most of the time there are very few messages in the network, however,




(a) Low-order interleaved memory.

(b) High-order interleaved memory.

Figure 5: Traffic pattern for FWA.

the maximum number of messages in the network at any time is much larger than in case of MATMUL, sometimes there are more than 2000 messages in the network. This clearly shows the bursty nature of traffic that can be generated by some shared-memory applications.

## 5.3 FFT

The execution times of FFT for different network parameters is shown in Fig 6. The execution time shows an improvement of about 25% when number of virtual channels are increased from 2 to 4. However, further increase in the number of virtual channels make only a small difference to the execution time. Increasing the number of flit buffers consistently improves the performance, but with diminishing returns. Increasing the number of internal input and output links to 2 or 4 makes only a small improvement.

The traffic pattern for FFT is shown in Fig 7(a) and 7(b). In case of low-order interleaved memory, the traffic is uniformly distributed across all the nodes. The small ridges in the otherwise flat plane is due to accesses to synchronization flags used in the barrier. In case of high-order interleaved memory, the traffic is concentrated to and from a small number of nodes.

For this application, most of the time the number of messages in the network are small, about 35 to 50, and the maximum number of messages at any time in the network is between 60 to 80. 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.

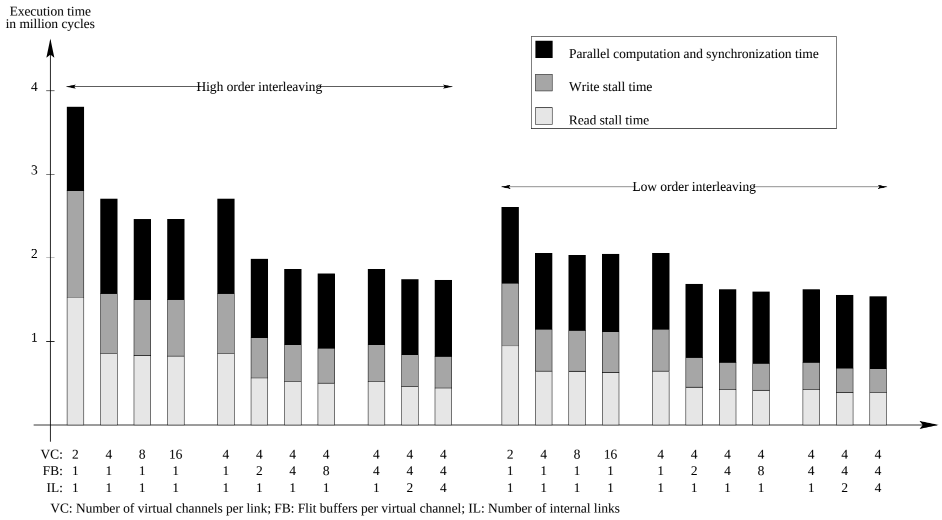
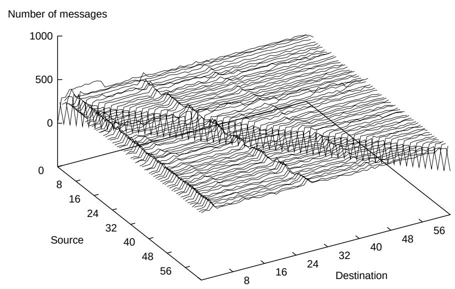
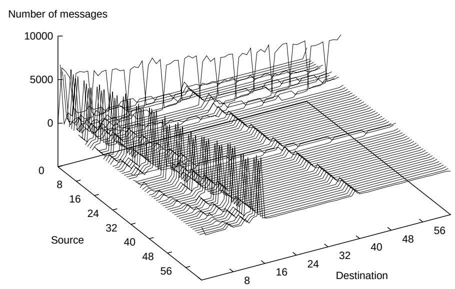



Figure 6: Execution time of FFT for various network parameters.




(a) Low-order interleaved memory.




(b) High-order interleaved memory.

Figure 7: Traffic pattern for FFT.

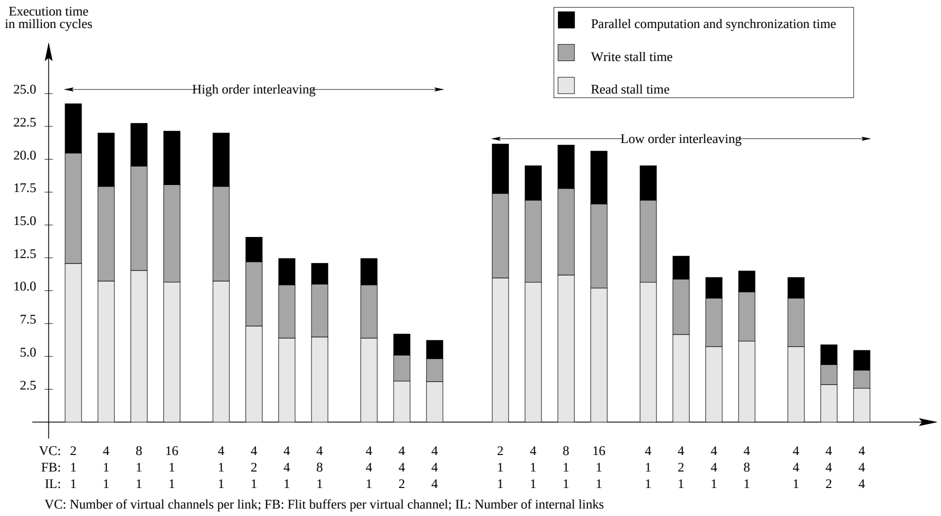


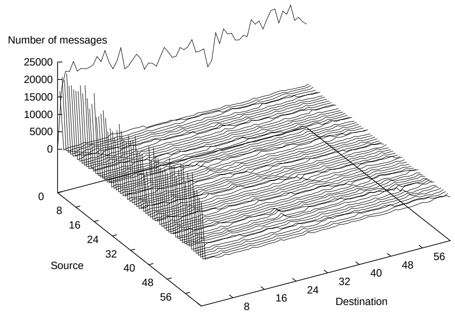

Figure 8: Execution time of MP3D for various network parameters.

## 5.4 MP3D

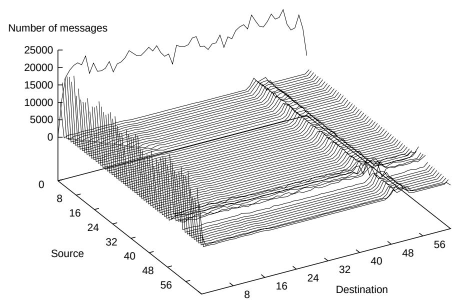
The execution times for MP3D for different network parameters are shown in Fig 8. Increasing the number of virtual channels from 2 to 4 decreases the execution time, however, the execution time for 8 and 16 virtual channels are more than the execution time for 4 virtual channels. The reasons for this unusual response is the segmentation of worms and increase in the number of messages, as explained earlier. 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 [17] 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. Increasing the number of flit buffers per channel up to 4 has considerable impact on the performance. In case of low-order interleaving, the execution time for 8 flit buffers is larger than the execution times for 4 flit buffers. The reason for this is explained in section 4.3. The increase in the number of internal links makes a large difference for this application since it helps to remove the congestion at nodes with contending requests.

A faster network also results in faster synchronization. This can be seen from the lower computation and synchronization time in Fig 8 for the network configurations that have 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.

Figure 9(a) and 9(b) shows the traffic pattern for the low-order and high-order interleaved memory. The high-order interleaving results in accumulation of traffic 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




(a) Low-order interleaved memory.




(b) High-order interleaved memory.

Figure 9: Traffic pattern for MP3D.

node 0. We used test-test-and-set with exponential backoff to acquire semaphore.

For this application, most of the time there are about 40 to 60 messages in the network, and the maximum number of messages in the network at any time varies between 140 to 180.

## 5.5 LU

The execution times of LU for different network parameters are shown in Fig 10. Increasing the number of virtual channels from 2 to 4 causes only a small decrease in the execution time. Increasing the number of virtual channels to 8 and 16 does not lead to any noticeable improvement, and sometimes even increases the execution time over 4 channels. Increasing the buffer size per channel from 1 to 2 causes appreciable decrease in execution time, however, any further increase in buffer size makes almost no difference. The number of internal links also has only a small impact on the performance. The improvements in execution time for this application are much smaller compared to other applications since the fraction of read and write stalls are much smaller.

The memory interleaving does not make any difference on the execution time in this case. 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, even in case of high-order interleaved memory, 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 number of nodes.

The traffic pattern for low-order and high-order interleaved memory is shown in Figs 11(a) and 11(b), respectively. The traffic is fairly distributed across all the nodes in case of low-order interleaving. In case of high-order interleaving, each node communicates with

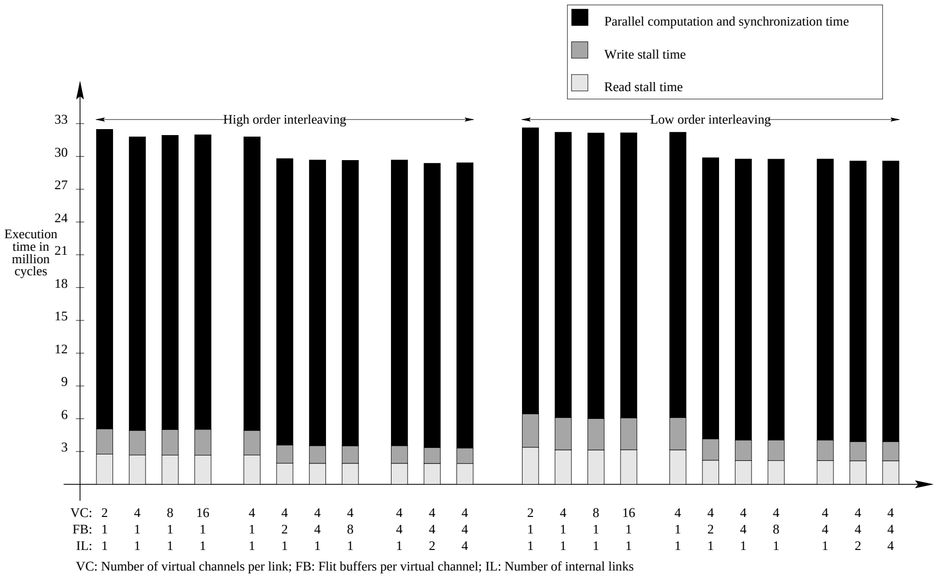


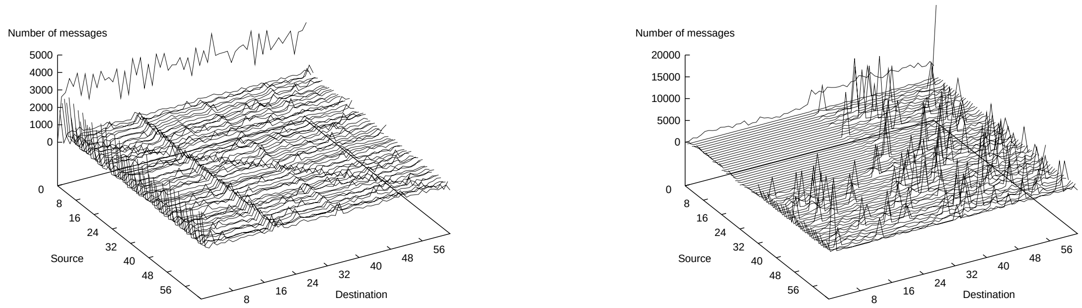

Figure 10: Execution time of LU for various network parameters.

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.

For LU, most of the time the number of messages in the network are very small, and at any time there are at most 85 to 125 messages in the network. This indicates that the message contention is not that severe for this application, which shows up as lower average message latency and wait times compared to other applications.

# 6 Conclusions

In this paper we evaluated the performance of a wormhole routed 2-D torus network using execution-driven simulation with some shared-memory applications. One of the 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 amout of buffer resource, it is necessary to properly balance the number of virtual channels and buffers per channel to obtain the best performance.




(a) Low-order interleaved memory. (b) High-order interleaved memory.

Figure 11: Traffic pattern for LU.

The number of links between the communication interface and the router has an impact on the performance when there is contention for a memory module or large number of messages are generated at a node in a burst. In case of high-order interleaving of shared memory blocks, increasing the number of internal links always helps by reducing the hot-spots at memory modules. Also, when the sharing characteristics of the application is such that large number of invalidations are generated, such as in case of FWA and MP3D, the larger number of internal links are beneficial. The distribution of shared memory blocks has tremendous impact on the execution time and the performance of the network. We have considered low-order and high-order interleaving of memory blocks without any user-defined placement of shared variables. For all the applications, except LU, the low-order interleaving gives better execution times and message latencies. In case of LU, both types of memory distribution perform equally well. The hot-spots in the network is reduced in low-order interleaving due to even distribution of traffic throughout the network. The advantage of low-order interleaving is more pronounced when the network is constrained with fewer virtual channels, flit buffers, and internal links.

The execution based evaluation of the network in this paper shows the impact of various network parameters and also points the ways to further improve the performance. Our measurements show that the utilization of the network and internal links 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 adaptive routing techniques to improve this situation and assess its benefit on the execution of shared-memory programs. We are also considering various message scheduling policies 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] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, "The Stanford DASH Multiprocessor," *Computer*, pp. 63–79, March 1992.

[6] K. Bolding and L. Snyder, "Mesh and Torus Chaotic Routing," Technical Report UW-CSE-91-04-04, Dept of Computer Science and Engineering, University of Washington, April 1991.

[7] 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.

[8] Intel Corporation,. *Paragon XP/S - Product Overview*, 1991.

[9] S. Chittor and R. Enbody, "Performance Degradation in Large Wormhole Routed Interprocessor Communication Networks," In *Proc. of the 1990 International Conference on Parallel Processing*, pp. I–424–428, August 1990.

[10] Y. M. Boura and C. R. Das, "Modeling Virtual Channel Flow Control in Hypercubes," In *Proc. of the First IEEE Symp. on High-Performance Computer Architecture*, pp. 166–175, Raleigh, NC, January 1995, IEEE Computer Society.

[11] Q. Yang, L. N. Bhuyan, and B. Liu, "Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," *IEEE Transactions on Computers*, vol. 38, no. 8, pp. 1143–1153, August 1989.

[12] J. Archibald and J.-L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Transactions on Computer Systems*, vol. 4, no. 4, pp. 273–298, November 1986.

[13] 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.

[14] 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.

[15] 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.

[16] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, May 1987.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.