Impact of CC-NUMA Memory Management Policies on the Application Performance of Multistage Switching Networks

Laxmi N. Bhuyan, Fellow IEEE, Ravi Iyer, Student Member IEEE, Hu-jun Wang, and Akhilesh Kumar

Abstract

In this paper, the impact of memory management policies and switch design alternatives on the application performance of cache-coherent non-uniform memory access (CC-NUMA) multiprocessors is studied in detail. Memory management plays an important role in determining the performance of NUMA multiprocessors by dictating the placement of data among the distributed memory modules. We analyze memory traces of several scientific applications for three different memory management techniques, namely buddy, round-robin & first-touch policies, and compare their memory system performance. Interconnection network switch designs that consider virtual channels and varying number of input buffers per switch are presented. Our performance evaluation is based on execution-driven simulation methodology to capture the dynamic changes in the network traffic during execution of the applications. It is shown that the use of cut-through switching with buffers and virtual channels can improve the average message latency tremendously. However, the choice of memory management policy affects the amount of network traffic and the network access pattern. Thus, we vary the memory management policy and confirm the performance benefits of improved switch designs. Results of sensitivity studies by varying switch design parameters, cache block size, and memory page size are also presented. We find that a combination of first-touch memory management policy and a switch design with virtual channels and increased buffer space can reduce the average message latency by as high as 70%.

Keywords

memory management, switch design, wormhole routing, execution-driven simulation, scientific applications, shared-memory multiprocessor.

This research has been supported by NSF grants MIP 9622740 and CCR 9810205.

L. N. Bhuyan, R. Iyer and H. Wang are with the Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, Email: (bhuyan,ravi,hwang)@cs.tamu.edu

A. Kumar is currently with Intel Corporation, 2200 Mission College Blvd, Santa Clara, CA 95052-8119, Email: akhilesh.kumar@intel.com

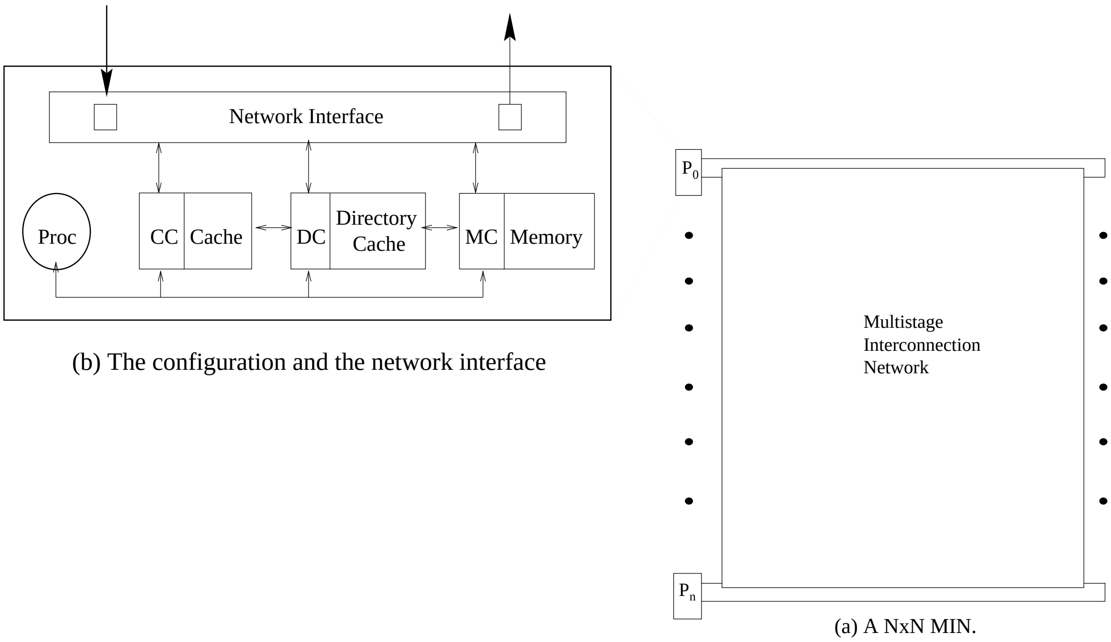
I. Introduction

Shared-memory multiprocessors provide a unified view of the memory for easy programming and are capable of providing significant performance benefits both for scientific and commercial applications. As a result, almost all the computer manufacturers have started marketing multiprocessor servers. While the small servers are based on shared-bus technology, medium scale multiprocessors with more than 16 processors employ sophisticated communication switches in a cache coherent non-uniform memory access (CC-NUMA) environment. Examples include SGI Origin, Convex Exemplar, and Sequent NUMA-Q machines. Design issues for faster communication time in CC-NUMA multiprocessors include the efficient use of processor caches, interconnection networks and memory management.

Cache memories are an integral part of shared memory systems to help avoid the large latency of memory accesses. The time to service a cache miss especially from a remote memory in a CC-NUMA system could be several orders of magnitude higher than the cache access time. Memory allocation or placement of data in a CC-NUMA multiprocessor is of significant importance to the performance since it determines the extent of remote memory accesses. To ensure high performance, it is essential for the operating system to employ memory management policies that allocate data in local memories intelligently, thus increasing data locality and eliminating or atleast significantly reducing the need for remote accesses. Previous work on memory management policies [9], [14], [13] was based on distributed shared memory multiprocessors without hardware cache coherence. The impact of memory management policies on CC-NUMA system performance has been recently studied in [18]. The study focuses at an application execution time level and assumes constant local and remote access latencies in its trace-based analysis. In this paper, we implement three different memory management schemes and evaluate their impact on the switch/network performance and the memory system performance. We study in detail the interference in the network due to data placement in the memory.

Performance evaluation of interconnection networks (INs) has been an active area of research for a long time [1], [2]. All these studies consider the network in isolation and are based on analytical models that assume simple traffic situations. These models do not consider different types of messages, bulky requests and invalidation traffic associated with cache coherence and synchronization in a shared memory environment. It is difficult to capture all the details of a network into a simple analytical or simulation model. On the other hand, numerous cache coherence studies of CC-NUMA architectures based on execution-driven simulation do not model the IN switches explicitly, rather they assume a fixed delay in the IN ignoring possible interference among various messages [6], [11]. Today's IN has become quite advanced with the introduction of techniques such as virtual channel and adaptive routing [4], [7], [8], [17]. The effect of all these advances in the INs has to be judged from the dynamic changes in traffic during the execution of applications. Execution-based evaluations for mesh interconnection networks have been reported recently [8], [17] to test the effectiveness of virtual channels and adaptive routing. In [17], the authors contradicted some results in [8] without understanding that their memory management scheme was different, and as a result, portrayed different results. Thus a detailed study on the impact of memory management on network performance is overdue.

We evaluate a multistage interconnection network (MIN) in this paper, similar to the one employed in Butterfly and Cedar multiprocessors. Our network is based on the same topology as Butterfly and Cedar, but has the following differences: (a) switching: we adopt wormhole routing, virtual cut-through switching and virtual channels, as employed in commercial multiprocessor switches [4], [7] instead of packet switching in Cedar and circuit switching in Butterfly, (b) cache coherence: we employ a directory-based cache coherence whereas Cedar and Butterfly have no hardware support for cache coherence and (c) organization: Cedar is a uniform memory access (UMA) multiprocessor which employs a forward network from processors to memories and a backward network from memories to processors. We employ a NUMA organization with one network, like Butterfly, that is used for both forward and backward (reply) messages. NUMA architectures offer good scalability for shared memory processing and a number of such machines are commercially available now. Our aim in this paper is



(b) The configuration and the network interface

(a) A NxN MIN.

Fig. 1. A MIN-Connected Multiprocessor

to re-evaluate the impact of the MIN switch design in a CC-NUMA environment employing various memory management policies.

The paper makes the following contributions.

- We present three realistic CC-NUMA memory management policies, namely *buddy allocation, round-robin allocation, and first-touch*, and characterize their effect on memory access patterns and remote access latencies for various applications.
- We present four different switch design alternatives called *simple wormhole (SWH)*, *buffered cut-through (BCT)*, *simple virtual channel (SVC)* and *buffered virtual channel (BVC)* switches. They differ in their buffer sizes and presence of virtual channels. We show that adding buffers and virtual channels improves the network and message latencies tremendously.
- We incorporate the three memory management policies and exact models of the four switch designs into a detailed application-driven simulation and measure the average read, write and invalidation latency in each stage of the network. For a CC-NUMA system using a directory-based cache coherence protocol, we observe that a combination of BVC switches and first-touch memory management performs the best for scientific applications.
- To illustrate the performance of different switch architectures and the impact of memory management policies, sensitivity studies with respect to switch size, switch delay, cache block size and page size are also presented and analyzed.

The rest of the paper is organized as follows. Section 2 describes the simulation environment, hardware parameters and application characteristics. Section 3 introduces the role of memory management in CC-NUMA systems and analyzes the impact of three memory management policies on memory access patterns and latencies. Section 4 presents the MIN architecture and various switch designs used in our evaluation. The impact of switch architectures on the waiting delay at various stages and its effects on stall time is also presented in this section. Section 5 presents the performance impact of memory management policies on the network performance. It also presents a sensitivity study to quantify the effect of switch design parameters and memory management parameters on system performance. Finally, Section 6 presents the conclusions of this study.

Parameter	Value
Number of processors	16
Shared memory size per node	512 Kbytes
Cache size	32 Kbytes
Cache line size	32 bytes
Set size	2
Cache access time	2
Directory access time	2
Memory access time	16
Switch delay	1, 2, or 4
Link width	16 bits
Flit length	16 bits
Message length	8 or 40 bytes
Page size	1 Kbytes
Interface delay	20

TABLE I
Simulation Parameters

II. Simulator Design

Our simulator is based on Proteus [3] which originally implemented an MIN using an analytical model. We have modified the simulator extensively to exactly model the MIN switches with wormhole routing and virtual channels. The system parameters used in the simulation are listed in Table I. The system architecture under consideration can be modeled as processor, cache, memory, network interface and the network, as shown in Fig. 1. This is a non-uniform memory access (NUMA) architecture where the memory is distributed among the processors, but all the memories share the same address space. The memory access time varies depending on the location of the memory word/block, hence the name NUMA. The interface provides services such as dividing the message into packets or flits, initializing header flit of every packet with necessary routing information, etc. Each node (Fig. 1b) contains a coherence controller, memory controller, directory controller and a reply controller, as described in [10].

We simulated a 16 node system using a MIN with 512KB of memory per node which is enough to hold the shared data of the applications. We chose a 16-node system to limit the simulation time, a larger system would need an appropriately larger data structure to get realistic results. Moreover, since our study focuses on the switch architecture instead of scalability, a detailed simulation of a 16 processor system is more appropriate than a high level simulation of a bigger system. The cache parameters used in the simulation are similar to the HP PA-8000. There is one level cache of size 32 KB per processor with a line size of 32 bytes, 2-way set associative organization and access time of 2 cycles. The switch design parameters are explained in section 4. A flit size of 16 bits was considered which is same as the width of the link. The switch latency or delay is considered to be 1, 2 or 4 processor cycles. The SGI Spider and Cavallino [7], [4] both have 4 cycles of latency. Additionally, we consider 1 or 2 cycle delays for comparison since our aim is to design high-speed switches. The interface delay (Fig. 1b) for protocol handling, etc. is 20 cycles, similar to DASH [10]. We assumed an infinite buffer at the interface, thus avoiding any deadlock problems. Messages are of two different lengths in our cache coherent multiprocessor with 8 and 40 bytes for control and data messages, respectively.

A. Cache Coherence And Synchronization

We implemented the full-map directory-based cache coherence protocol [5] with some modifications 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 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.

We have modified the cache coherence protocol where the cache controller detects if a message has arrived out-of-order and holds it for later service. Such a situation is possible due to the existence of virtual channels. The synchronization method used in our simulations is based on spin-locks using test-and-set operation with exponential backoff [12]. Exponential backoff improves the system throughput by enforcing delays between consecutive test-and-set operations based on an initial constant k that increases geometrically for every subsequent attempt. 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 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 FFT and MP3D[15]. The network latency also plays an important role on the synchronization overhead. In this paper, we do not discuss synchronization overhead associated with any of the simulations, but concentrate only on read/write and coherence requests to data items.

B. Benchmark Applications

We have selected some numerical applications for evaluating the network performance in a cache-coherent shared-memory environment. These applications are multiplication of two 2-D matrices (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 description of the applications below also present the input data sizes for the 16-processor simulation.

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 matrices is transposed to avoid conflict misses.

For Floyd-Warshall's algorithm, we used a graph of 256 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 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 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 processor) are allocated contiguously, and false sharing and line interference are 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 arrays 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 the remaining $\log_2 P$ stages, every data point is shared by two processors. During these stages, instead of using two separate input and output arrays, we interleave these arrays to avoid conflict misses.

MP3D is a three-dimensional particle simulator used in rarefied fluid flow simulation. We used 25000 molecules with the default geometry provided with SPLASH [15] 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 only for 5 time steps to limit the simulation time. 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.

III. Characteristics of Memory Management Policies

Several memory management policies have been suggested for a distributed shared memory system [9], [14], [13], but not much work has been done for CC-NUMA architectures. The only notable work, reported in [18], examines the performance of static and dynamic allocation schemes at the application execution time level. Our aim is to study memory access patterns and present a detailed analysis of its impact on the network performance. We consider the following memory management techniques in this study.

- **Buddy Allocation:** In this allocation scheme, the data is allocated contiguously based solely on the requested data size and the available memory space in the system. The entire shared memory space is divided into buckets each with memory capacity increasing by a power of 2. The starting address of the requested data chunk is determined by performing a best-fit search on the buckets with sufficient capacity. Such an allocation scheme allows a low overhead during data allocation. This scheme also provides a suitable method for maintaining free lists of unallocated memory segments and is widely used in uniprocessor and small multiprocessor systems. However, as we will see, the CC-NUMA performance is poor since the scheme saturates a particular memory with multiple accesses from several processors. Data accesses are directed towards a selected few nodes in the system, causing severe memory and network contention.
- **Round-Robin Allocation** In this allocation scheme, pages of data are allocated in a round-robin fashion across the nodes. A page i is allocated to memory i *Modulo* P, where P is the number of memory modules in the system. The distribution of the pages is extremely uniform thus smoothening out the traffic generated in the network and in most cases making sure that simultaneous access to different data from different processors are interleaved among the different nodes. This scheme is used in DASH [10] and commercial CC-NUMA multiprocessors. However, the scheme does not place data intelligently enough to reduce the number of remote accesses. Since the distribution is regular in most scientific applications, many processors tend to access multiple pages on the same node in a given time interval. As we will see in section 4, requests are queued up at the network interface.
- **First-Touch Allocation:** In this allocation scheme, a page is located near the processor that first accessed the page. In other words, at the time of the first page fault, the page is allocated to the local memory of the faulting processor assuming that it will be used repeatedly by the same processor. A well-written application can exploit the use of this scheme and certainly gain in performance. On the other hand, poorly written applications (eg: initializations performed at

Application	PID	Buddy				Round-Robin				First-Touch			
		0	1	2	3	0	1	2	3	0	1	2	3
FWA	0	0.00	0.00	0.00	1.00	0.24	0.22	0.24	0.30	0.95	0.02	0.02	0.01
	1	0.00	0.00	0.00	1.00	0.23	0.23	0.26	0.27	0.03	0.94	0.02	0.01
	2	0.00	0.00	0.00	1.00	0.24	0.25	0.24	0.27	0.03	0.02	0.93	0.01
	3	0.00	0.00	0.00	1.00	0.26	0.23	0.23	0.29	0.03	0.02	0.02	0.93
MATMUL	0	0.00	0.00	0.00	1.00	0.23	0.24	0.26	0.27	0.53	0.04	0.43	0.00
	1	0.00	0.00	0.00	1.00	0.25	0.22	0.26	0.28	0.10	0.47	0.00	0.43
	2	0.00	0.00	0.00	1.00	0.27	0.26	0.24	0.24	0.46	0.00	0.51	0.03
	3	0.00	0.00	0.00	1.00	0.28	0.28	0.24	0.21	0.03	0.43	0.07	0.47
LU	0	0.00	0.00	0.00	1.00	0.30	0.23	0.24	0.23	0.97	0.02	0.01	0.00
	1	0.00	0.00	0.00	1.00	0.31	0.21	0.24	0.23	0.99	0.01	0.01	0.00
	2	0.00	0.00	0.00	1.00	0.30	0.22	0.24	0.24	0.98	0.02	0.01	0.00
	3	0.00	0.00	0.00	1.00	0.30	0.22	0.24	0.24	0.99	0.00	0.01	0.00
MP3D	0	0.01	0.00	0.40	0.59	0.26	0.25	0.25	0.24	0.23	0.09	0.17	0.52
	1	0.01	0.00	0.40	0.59	0.26	0.25	0.25	0.25	0.20	0.10	0.18	0.52
	2	0.01	0.00	0.40	0.59	0.26	0.25	0.25	0.25	0.19	0.09	0.20	0.52
	3	0.00	0.00	0.41	0.59	0.25	0.25	0.25	0.25	0.18	0.09	0.18	0.55
FFT	0	0.00	0.00	0.00	1.00	0.26	0.26	0.24	0.24	0.88	0.07	0.05	0.00
	1	0.00	0.00	0.00	1.00	0.26	0.26	0.24	0.24	0.06	0.89	0.00	0.05
	2	0.00	0.00	0.00	1.00	0.26	0.26	0.23	0.24	0.08	0.00	0.85	0.08
	3	0.00	0.00	0.00	1.00	0.26	0.26	0.24	0.24	0.00	0.05	0.06	0.89

TABLE II
VARIATIONS IN MEMORY ACCESS PROBABILITIES

a single processor before main parallel computation begins) will locate pages incorrectly based on the first access and cause several remote memory accesses later.

In order to understand the behavior of the above memory management policies, we observed how the memory requests are distributed in our simulated CC-NUMA architecture for each application. Table II shows the probability distribution of the data accesses in a 4-processor system under different memory management policies for all applications. The processor id (PID) that generates the requests is given in the second column. We observed similar results for all applications on a 16-node system but chose to show the 4-node results for brevity. Consider the FFT memory distribution probabilities. From the table, it is clear that when the buddy memory management policy is used, the entire chunk of shared data gets allocated in a single memory module (node 3), so all processors access memory 3 all the time. When we switch to the round-robin memory management policy we see a uniform distribution of data over all memory modules with an approximately 25% $(1/P)$ probability including the local memory. With first-touch, this local memory access probability increases to as high as 89% for the FFT application. Thus the number of remote requests and responses that traverse the network is reduced tremendously. Such an improvement in localizing data accesses will reduce the cache miss latency and provide considerable improvement in stall-time performance.

While the above probabilities characterize spatial distribution of memory accesses, the performance of an application is also dependent on temporal locality of a processor's request. Let us define *think time* of a processor as the time a processor spends in execution before sending a memory request. If the cache is included in a processor model, think time is the time between when a cache miss is satisfied and the next one is generated. This parameter does not include the memory access time (local or remote) and is independent of the memory management policy. Let us define $inter-arrival$

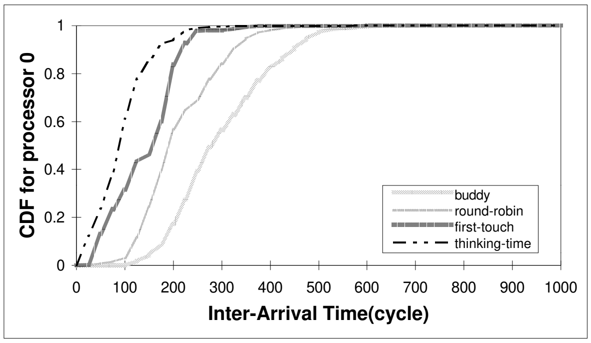


Fig. 2. FFT Memory Access Distribution

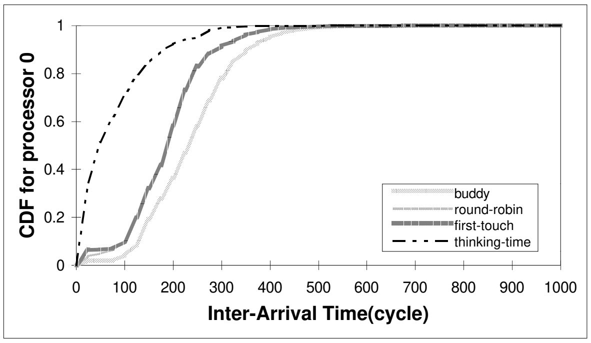


Fig. 3. MP3D Memory Access Distribution

time as the time between two cache misses including the memory and network access time. If all the data were placed in the local memory, then there is no network access time involved and *inter-arrival time* $=$ *think time*$+$*memory access time*. On the other hand, with a bad memory management policy, the inter-arrival time is higher due to access to the remote memory. In the worst case, *inter-arrival time* $=$ *think time* $+$ *memory access time* $+ 2$ *network latency times*.

Figures 2-6 present the cumulative distribution function (CDF) of the inter-arrival time for different memory management policies. Each figure consists of four curves; one representing the workload think time and three others representing the distribution of inter-arrival time for the memory management policies. A point (X,Y) on the curve denotes the probability Y that the inter-arrival time (IAT) is X or lower ($P(IAT \leq X) = Y$). Figure 2 clearly shows the difference between latencies observed for each memory management policy for the FFT workload. On the extreme left, we see the think time. The lowest latency for memory access is provided by first-touch with an inter-arrival time lower than 200 processor cycles during 80% of the parallel execution time. On the other hand, round-robin provides a latency of 300 and buddy provides a latency of approximately 420 for the same period of time. Thus first-touch improves the memory access latency by approximately 51% and round-robin improves by

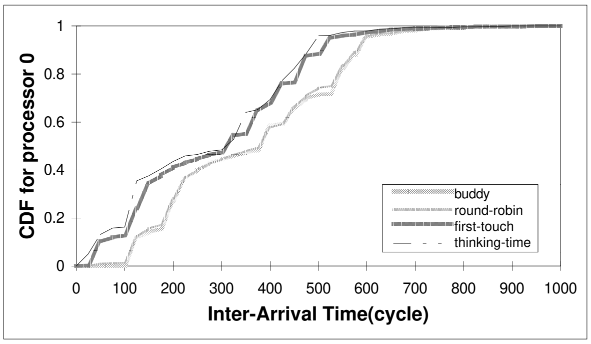


Fig. 4. LU Memory Access Distribution

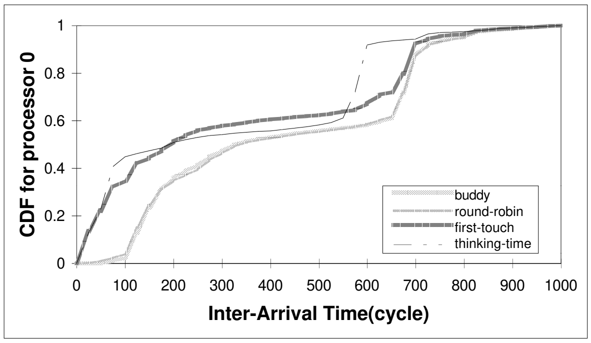


Fig. 5. FWA Memory Access Distribution

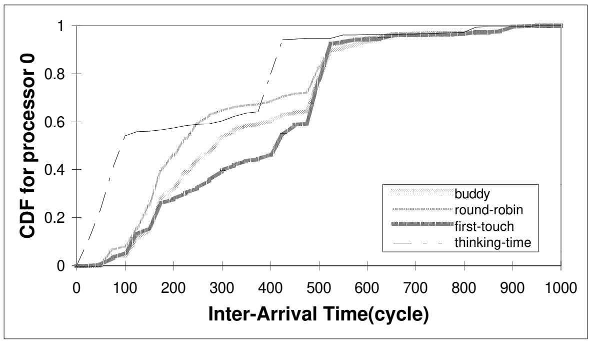


Fig. 6. MATMUL Memory Access Distribution

25% over the buddy memory management policy.

While observing the memory access patterns, we find that FFT, MP3D and LU have temporal access patterns that fit CDFs of known distributions like exponential or hyper-exponential distribution. However FWA and MATMUL applications exhibit jumps in the inter-arrival times and are difficult to characterize. For matrix multiplication, Figure 6 shows a probability of approximately 60% that the think time is less than 100 and a probability of close to 40% of a think time higher than 400. This jump in think time is due to the frequent cold misses that occur initially while reading row and column elements of the respective input matrices. Since the misses occur in the inner-most loop of the application, the think time between the misses is low (≤ 100). Once these cold misses are served and the rows/columns are read into the caches, subsequent computations requiring these elements are cache hits. From then on, cache misses become infrequent and occur only while performing writes to elements of the result matrix. Since these misses are in the outer loop of the program, the think time is high (≥ 400).

IV. Performance Evaluation of Switch Architectures

In this section, we present four different switch design alternatives for the multistage interconnection network (MIN) and evaluate their impact on performance at two different levels. We start by measuring the average message latency for read, write and invalidation requests of the applications in subsection 4.2. The average message latency is obtained by dividing the total time taken for these types of messages by the number of such messages throughout the execution of the program. The next subsection measures the impact of the switch architectures directly on the application performance. The stall time of each processor due to memory requests is measured and analyzed. During this evaluation, we use a page-wise round-robin policy for the placement of data in the memory.

A. Switch Design Alternatives

The schematic of the network is shown in Fig. 7. It is a multistage interconnection network (MIN) employing 2x2 switches. In general, an NxN MIN consists of $\log_2 N$ stages of 2x2 switches with N/2 such switches per stage. The interconnection between the stages is a perfect shuffle. When a processor generates a read or write request, different actions are taken according to the directory-based cache coherence protocol and depending on the status of the block in the cache, as explained in section 2.1. For example, if it is a read miss and the local memory is the home node, the request is satisfied locally. If the home is a remote node, the request is sent over the network to that node. At the remote node interface, the directory is checked and the data is supplied to the requester over the network again. The paths taken for the forward request and the reply message are different, as shown by bold lines in Fig. 7 for processor P0 and memory M15.

We use wormhole routing and virtual cut-through switching techniques. In wormhole routing, the messages are divided into small flits, typically 8-32 bits long which is the same as the link width between two adjacent switches. In case of blocking, the worm stands on the path until the block is removed. In virtual cut-through, buffers are provided in the switch so that a message can be completely buffered when it is blocked. However in our case, a message is buffered to the extent allowed by the buffer size. Part of the message may be buffered and the other part may stand like a worm. There are two types of messages that are transmitted over the MIN. One is a *control* message that consists of read, write and invalidation signals that are short and 4 flits long. The *rdata* and *wdata* messages supply the block to the requesting cache and are 20 flits long for a cache line size of 32 bytes. Wormhole routing or virtual cut-through will be better than packet switching because it will save 20 cycles per switch without having to buffer the large data packets.

A simple wormhole (SWH) 2x2 crossbar switch is shown in Fig. 8.a. The buffer space needed per input and output for wormhole routing is only one flit that is essential for transmission over the links. Once the flit arrives, the arbitration starts and the connection is made. Such a simple crossbar

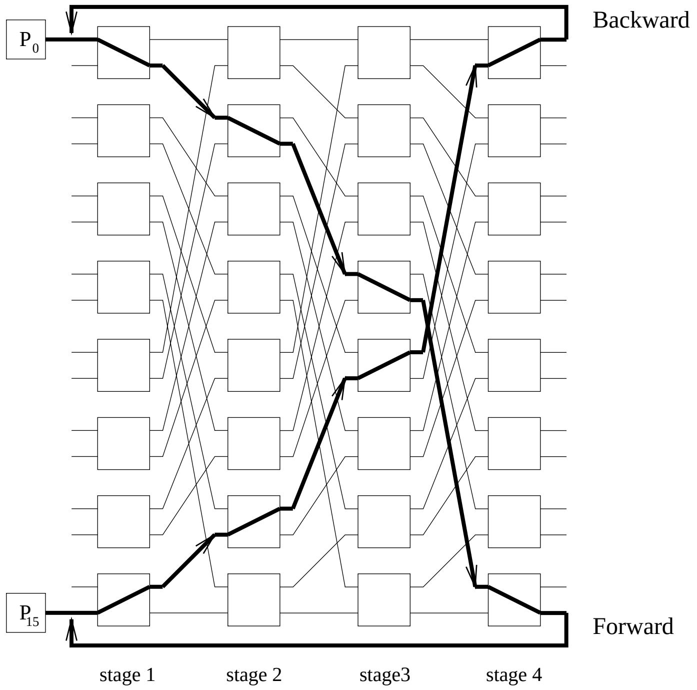


Fig. 7. A 16×16 MIN using 2×2 switches

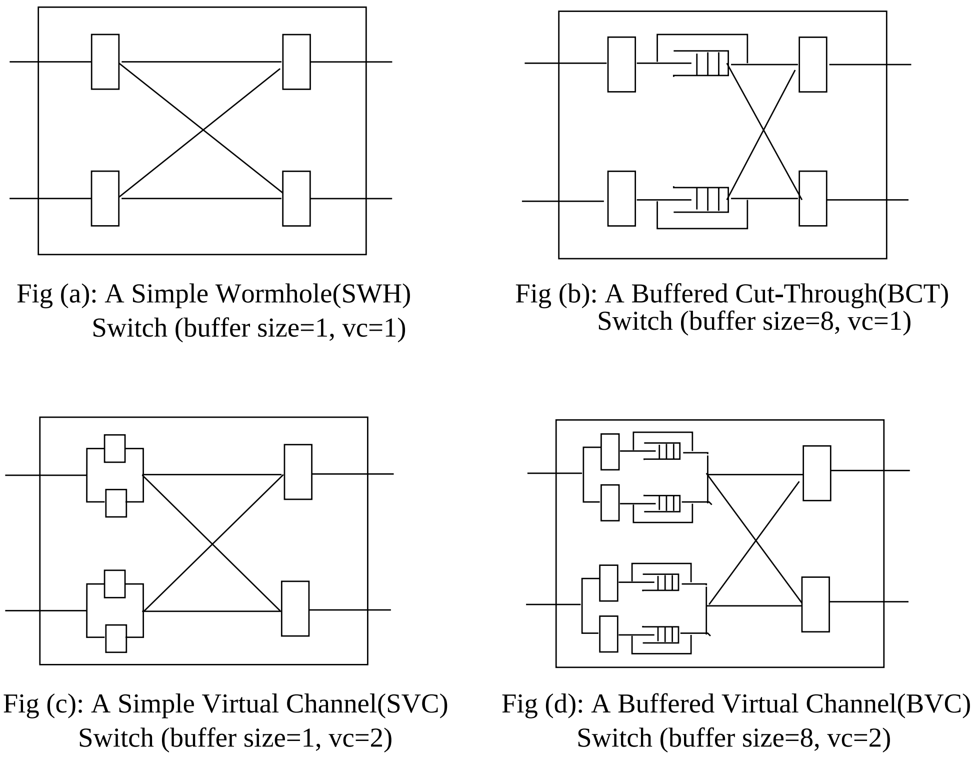


Fig. 8. Four Designs for a 2x2 Crossbar Switch

arbiter takes one cycle arbitration time, however, the grant signals are modified to ensure continuity of transmission due to wormhole routing. The input synchronization takes one to two cycles, and it takes one more cycle for transmission. As a result, the switch delay is about 4 cycles, which is similar to the time taken in SGI Spider and Intel Cavallino switches [4], [7]. Many times the requests arrive in bulk. For example, in case of a write miss on a block, the interface/coherence controller sends invalidation signals to processors having a copy of the block one after another. In such situations, providing some buffer in the switches will be helpful. We study the effect of providing a buffer size of 8 flits and 32 flits so that a control message or a data message can be held entirely in the switch if required. However, if the output is free, the arriving message bypasses the buffer. Note that the output buffer is still one flit. We call such a switch a buffered cut-through (BCT) design and is shown in Fig. 8.b.

Virtual channels (VC's) are used in wormhole networks to avoid deadlocks and to improve link utilization and network throughput. Fig. 8.c shows such a simple virtual channel (SVC) design with two VC's. Whenever there is blocking of a message, another message can be routed through the extra VC. In this situation, the arbiter consists of two simple 4 to 1 crossbar arbiters. When the number of inputs is more (6 inputs with 4 VC's or buffers per input which is equivalent to 24 possible inputs per output in Cavallino and Spider [4], [7]), two-stage arbiters are needed to keep arbitration time within limit. However, we have no such problem with a simple 2x2 switch. Each of the virtual channels can again have space for multiple flits. Flits from only one message can occupy the buffers at one virtual channel at a time. There are no buffers on the output side so that an unnecessary delay can be avoided. We call such a design a buffered virtual channel (BVC) switch, as shown in Fig. 8.d. As denoted in Fig. 8, the switch designs can be represented in terms of two parameters, buffer length (B) and number of virtual channels (VC). For SWH, B=1 and VC=1; for BCT, B=8 or 32 and VC=1; for SVC, B=1 and VC=2; and for BVC, B=8 or 32 and VC=2.

The BVC switch is quite complicated because there can be four different messages per input competing for an output port at a time. We designed a modified version of round-robin arbitration policy to reduce the message latency. Here the virtual channel that used the link in the previous cycle gets priority if it is still transmitting the flits from the same message, otherwise the round-robin policy is used. When the last flit of a message is sent, the round-robin policy avoids starvation on other channels. The crossbar output selection and conflict resolution is a two-step process. In the first step, the input channel that has a message to send, generates a request for the output channel. Multiple input channels can request the same output channel in this phase. This conflict is resolved in the second step. A similar arbitration policy is used in the Intel Cavallino switch [4].

B. Impact on the Average Message Latency

Messages encounter delays at various stages (1 through 4) of the network, shown in Figure 7. In addition, there is a delay at the interface (called stage 0) to pump the message into the first stage of the network. The purpose of this section is to analyze the waiting delays at different stages for various applications and measure the effect of the improved switch designs, presented in section 4.1. Only round-robin (RR) memory management is considered in this section.

We present the results for different applications under two groups. The first group consists of MATMUL, LU and MP3D whose results are presented in Figs. 9, 10 and 11 respectively. The average message latencies for read, write and invalidation are plotted separately, as measured from the simulation. For each type of message, the latency is plotted for simple wormhole (SWH), buffered cut-through (BCT), simple virtual channel (SVC) and buffered virtual channel (BVC) switch architectures. Buffer sizes of 8 and 32 flits are considered for BCT and BVC switches. Also, three types of messages are considered. They are read, write and invalidation requests, as noted on the x-axis.

Consider the first bar for read requests in Fig 9. Starting from the bottom, we first show the transfer time (fw_d) over the network for the forward message which is the read request. This is the time without considering any waiting at switches. The next latencies are the waiting times that occur at

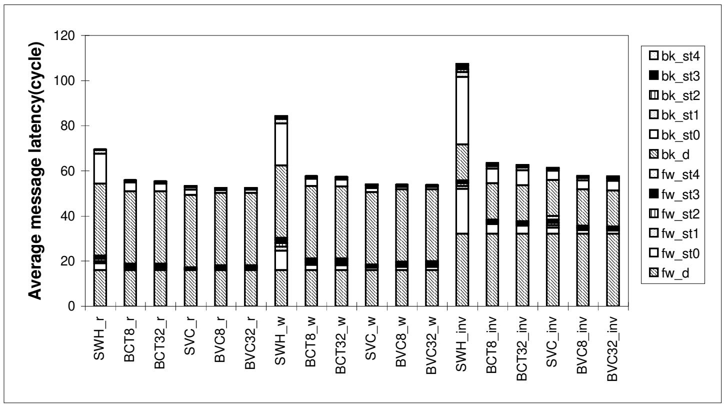


Fig. 9. Average Message Latency for MATMUL Application

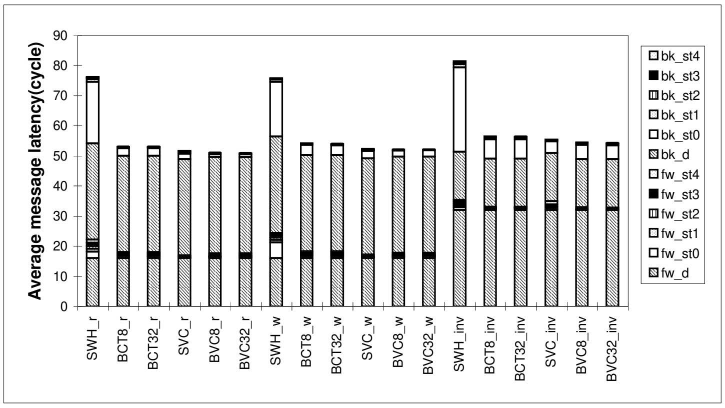


Fig. 10. Average Message Latency for LU Application

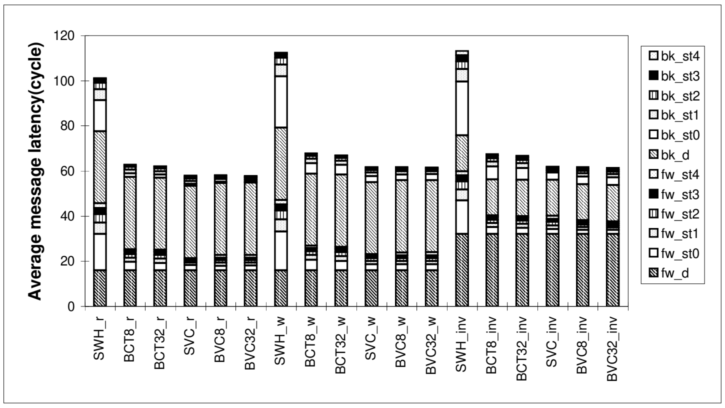


Fig. 11. Average Message Latency for MP3D Application

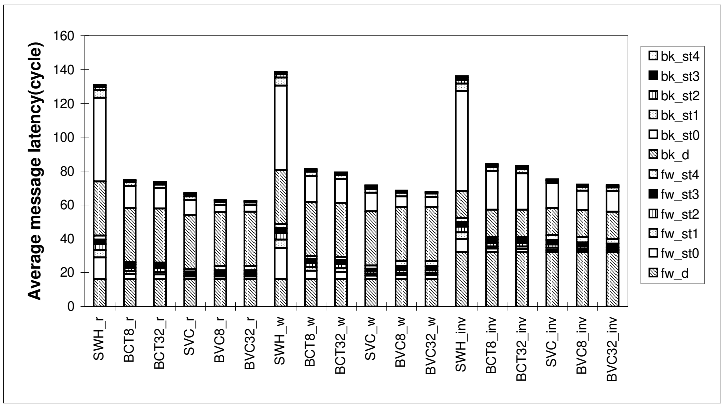


Fig. 12. Average Message Latency for FFT Application

stages 0 through 4, as denoted by fw_st0 through fw_st4. Stage 0 is the interface and stages 1 through 4 correspond to the 4 stages of the MIN from left to right, as marked in Fig. 7. On the upper half of the same bar, time taken for the reply (backward) message over the network is also plotted. Since a data block consists of 20 flits, the transfer time is much bigger, as shown by bk_d. The next field (bk_st0) indicates the time taken at stage 0 which is at the interface. This does not include times for memory access, directory access or protocol processing. Figs. 12 and 13 present the average message latencies for the second group of applications, namely, FFT and FWA. In these two figures, one can notice a huge waiting delay at stage 0 (interface) for the backward messages. A similar situation was also observed in Cedar network [16] where there was a large delay at the input of the backward network. In our case, the same network is used both for forward and backward requests and we use wormhole or cut-through switching instead of packet switching in Cedar. Even then, we notice the same phenomenon. As explained later, we will see that this phenomenon occurs due to the round-robin memory management policy. It disappears when a first-touch policy is adopted.

Figs. 9 through 13 also show the waiting times at different stages for all requests. They appear to be negligible, meaning that there isn't much interference between various messages in the network. Read, write and invalidation messages are considered, as denoted by suffix *_r*, *_w*, and *_inv* respectively, on the x-axis. The write latency consists of both the forward write miss request and backward data block supply only. It does not include time taken for invalidation. The latencies due to invalidation requests and their acknowledgment or update (in case of a dirty block) are counted in the last set of bars in Figs. 9 through 13. The forward invalidation requests originate from the home memory and are sent to all nodes that have a copy of the block. From the transmission delay (fw_d), it may be observed that there are more than one copy of the block because the time taken is more than the corresponding delays for a read or write request. Also, there is a large delay at the interface for the backward message. Since acknowledge messages are short, the delay is due to the presence of dirty copies in remote nodes which have to be updated at the home memory. Also, like read data, there seems to be an accumulation of these dirty blocks at the same node at a time.

For almost all applications, we observe a big delay at stage 0 or the input of the network. In order to understand why there is such a big delay at the input, we need to understand the operation of the network interface in detail. The interface, shown in Fig. 1b, provides a large buffer space sufficient for holding all incoming and outgoing messages. Whenever there is space in the output transmit buffer, the large interface buffer is bypassed. The average queue length for the forward and backward requests at the interface buffer is shown in Table III for different applications. A big difference in

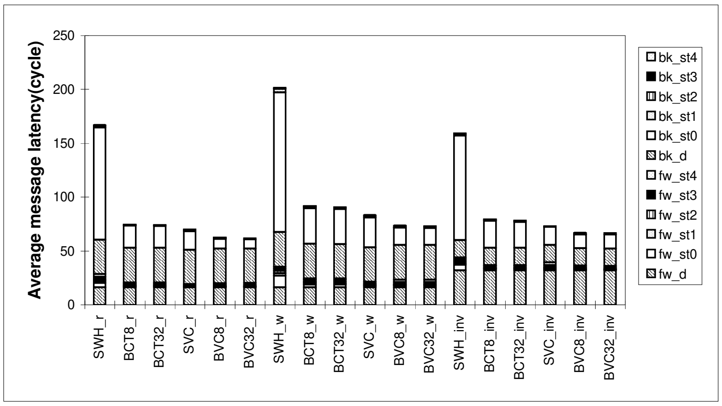


Fig. 13. Average Message Latency for FWA Application

Application	Message Type	Queue Length
FWA	forward	0.019304
	backward	0.146970
FFT	forward	0.066796
	backward	0.202756
MATMUL	forward	0.012961
	backward	0.022165
LU	forward	0.012619
	backward	0.032083
MP3D	forward	0.110048
	backward	0.120732

TABLE III
Average Queue Length at Network Interface

queue length between the forward and backward messages may be noticed in case of FWA and FFT applications. More importantly, we observed that these backward messages occur in burst not being uniformly spread across the execution time. It is also seen that memory requests from processors are directed to a particular memory (or page) at a certain time. Hence, at a time, the memory or interface has to supply the data blocks to various processors one after another. Since the memory access rate (16 cycles) is smaller than the network transmission time of about 30 cycles (without any interference in the network), these messages are queued up temporarily in the interface buffer.

The above situation is caused due to the round-robin memory management policy of page allocation and due to the behavior of applications in that the data items are either read/written from/to one page at a time, or different pages are accessed but wrap-around to the same memory module. It is because a data size of a power of 2 is distributed over a number of memory modules which is also a power of 2. Given such an access pattern, our aim is to reduce this delay at the interface by designing a smoother pipeline in the switch architecture. We also consider the impact of the other two memory management policies and analyze their impact on the performance of these switch architectures in the next section.

The next bar in Figs. 9 through 13 shows a drastic reduction in the input delay in buffered cut-

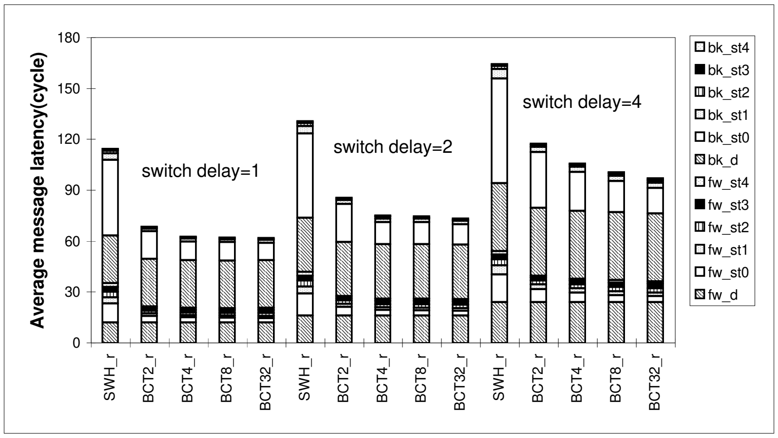


Fig. 14. Average Read Message Latency for FFT Application

through (BCT) architecture (Fig. 8b) where we provide a buffer of 8 flits at the input of the 2x2 switch. Increasing the buffer size to 32 flits, which can hold a complete data message (20 flits), does not improve the performance any further. The same gain can be obtained by providing two virtual channels per switch through a simple wormhole virtual cut-through (SVC) design, shown in Fig. 8c. With a buffered virtual cut-through (BVC) design of Fig. 8d, the latency can be cut down a little further, but there is almost no difference in results for buffer lengths of 8 or 32 flits.

In order to understand why there is such a big improvement in latency between buffer sizes 1 and 8, but no improvement between 8 and 32, we performed another set of experiments by varying the buffer size. Fig. 14 shows the average read message latency for FFT application for SWH and BCT architectures with buffer size 1, 2, 4, 8 and 32. We also vary the switch delay to be 1, 2 and 4 cycles. Since a header is blocked for the switch delay number of cycles at a switch, providing a buffer size equal to the switch delay will ensure a continuous flow of the message from the interface and will avoid waiting there. Providing any more buffer than the switch delay will not increase the performance. This is true assuming that there is no interference between various messages in the network. It is noticed in Fig. 14 that most of the gain is achieved when buffer size equals the switch delay. Improvement after that is minimal, and is due to a little interference that is still present in the network.

C. Impact on Application Stall Time

The average message latency metric quantifies the delay involved in a single message to the remote node. It accurately represents the interference between the messages in an application, but does not say anything about how these messages affect the program execution or the total stall time by a processor. Stall time is the total amount of time a processor waits for read/write operation. Apart from the cache miss latencies, it also partly represents the amount of interprocessor communication involved in an application. Next, we present read, write and invalidation stall times for different applications as a function of switch delay and link width. Since the BVC architecture performs the best, we fix the switch architecture with two virtual channels and a buffer size of 8 flits in this section. Again, only round-robin (RR) memory allocation is considered.

The current switch delay in the commercial machines is about 4 switch core cycles (switch core operates slower than the CPU) for a 6x6 crossbar [4], [7]. With a 2x2 switch, we could obtain a faster switch because it takes much less time to arbitrate. We assume the switch core frequency to be the

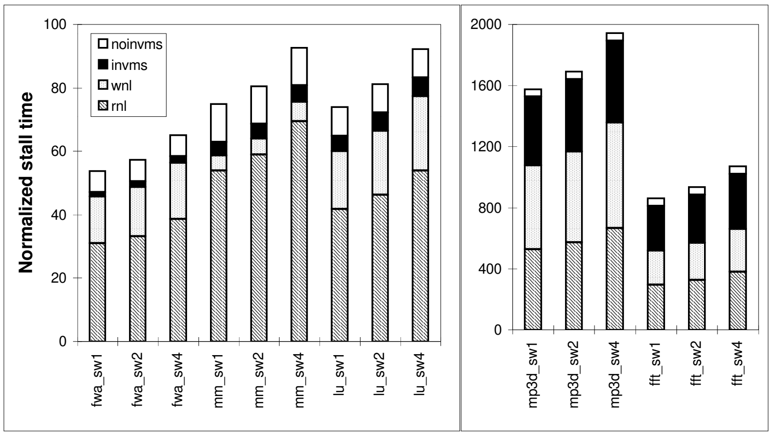


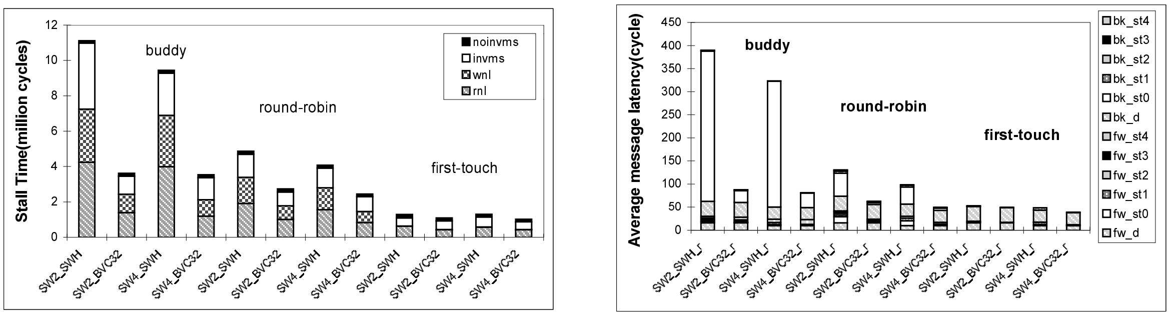
Fig. 15. Normalized Stall Time in 16-Node MIN Architecture

same as the CPU clock frequency and derive results with a switch delay of 1, 2 and 4 cycles in this section. The flit size is kept same as the link width (16 bits) in order to transmit one flit in a single cycle.

Stall time can be divided into several components based on different system components. However, we only present the network stall time and memory stall time separately. The read network latency (RNL) is the total time spent in the network during execution of a program to transfer the read requests and their replies. Similarly, the write network latency (WNL) is the total time spent in the network during program execution to transfer the write requests and their replies. It does not include the invalidation time, if any. Memory Service Time is the time taken for both read and write requests to be serviced by the memory controller thus generating a reply. Further we would like to address the invalidation overhead. Thus we divide the requests that reach memory into those that require invalidations and those that do not. These are called memory service time with invalidations (INVMS) and memory service time without invalidations (NOINVMS), respectively.

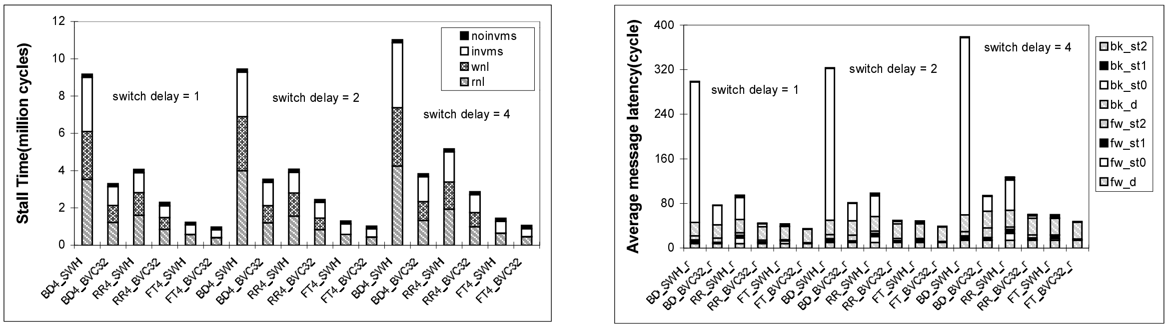
Figure 15 shows the performance variation of RNL, WNL, INVMS, and NOINVMS stall time for various applications with switch delays (sw) of 1, 2 and 4 cycles. The variation is not very significant compared to the overall stall time. All the results have been normalized with their computation times, so that the y-axis represents the proportion of time spent on stalling for different applications. Normalized Stall Time is 1000 times the ratio between stall time and parallel computation time of the same application. For FWA, LU and MATMUL, we use the scale on the left side in the figure. The stall times of MP3D and FFT applications are high, so they are plotted on a different scale. The stall times of applications like FWA, LU and MATMUL are quite low because they are mostly shared read kind of applications. On the other hand, applications like MP3D and FFT spend as high as 50% of their execution times in waiting for read/write/invalidation messages.

It may be noticed from Fig. 15 that the network stall time increases by about 10% with increase in the switch delay from 1 to 4 cycles. For FWA, MATMUL and LU, the read network stall times dominate because they involve mostly shared read data. For MP3D and FFT, the stall times are distributed among read, write and invalidation messages because these applications have a higher percentage of write sharing. We also observe that the invalidation overhead is significant. We can reduce the invalidation and write times by one network latency by incorporating dirty forward where the processor having a dirty copy sends the block directly to the requester instead of through the



(a) Application Stall Time (b) Read Network Latency

Fig. 16. Effect of Varying Switch Size on FFT



(a) Application Stall Time (b) Read Network Latency

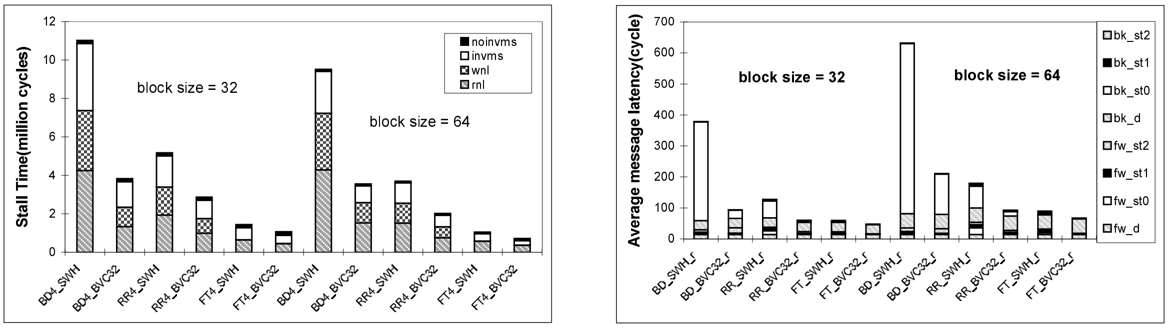
Fig. 17. Effect of Varied Switch Delay

directory update, like in DASH [10]. The invalidation overhead is also highly dependent on the network design. To reduce this overhead we need to employ techniques such as broadcasting and multicasting to perform many invalidations simultaneously. The faster the network, the faster the invalidation requests and replies are sent or received. In Figures 9 through 13, we showed that the invalidation delay reduces tremendously with increased virtual channels and buffer space.

V. Impact of Memory Management on Network Performance

In the previous section we analyzed the impact of the switch design on the application performance. In Section 3, we presented the role of memory management in determining the memory access pattern. In this section, we present a deeper analysis of the average network latency and the application stall time with varied memory management policies. We use the SWH and BVC32 switch architectures for our investigation below because they represent extreme examples of switch designs. We observe similar results for most applications but chose to present only the FFT application results for brevity.

Figure 16 presents a comparison of changing switch sizes, switch designs and memory management policies for 2×2 (sw2) and 4×4 (sw4) switches in terms of stall time and read network latency. The first thing we notice is that the first-touch (FT) memory management policy performs much better that buddy and RR policies. We observe that the improvement from SWH (low-end switch) to BVC32 (high-end switch) is very significant for buddy and round-robin policies. The BVC32 switch improves all components (reads, writes and invalidations) of the stall time. Figure 16 provides a stage by stage breakdown of the delays in the MIN. It is seen that the switch architecture mainly helps in reducing the backward stage 0 waiting delay by providing alternate routes through different virtual channels. We can see that increasing the switch size reduces the average message latency for all memory management policies and switch designs. Note that when a 4×4 switch is used in a 16-processor system, the number of stages in the network is reduced to 2, so the stall time improves. The delays fw_st3, fw_st4, bk_st3 and bk_st4 are not present in the columns of Fig. 16b for the 4×4 (sw4) switch results. The effect of



(a) Application Stall Time (b) Read Network Latency

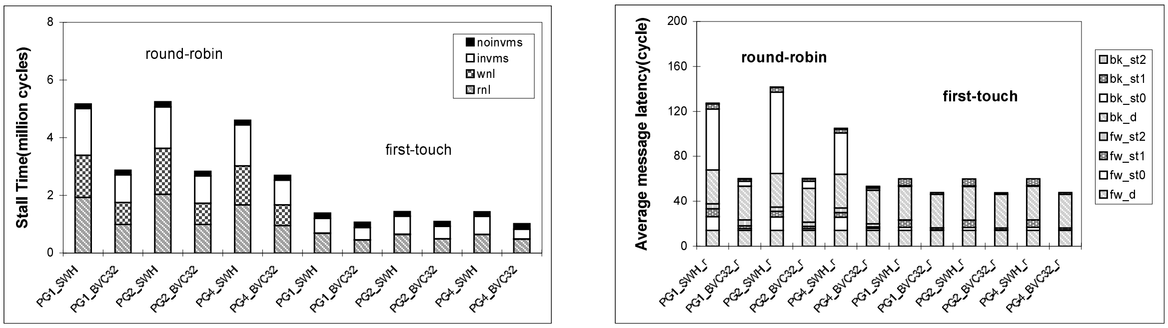
Fig. 18. Effect of Varied Block Size

virtual channels on buddy allocation and round-robin is more noticeable than on first-touch for both 2×2 and 4×4 switches. With first-touch, we see negligible improvement for switch size 2×2 and moderate improvement for switch size 4×4 due to less contention in the former. A switch delay of 2 cycles was assumed both for 2×2 and 4×4 switches. The switch delay is the number of cycles each switch takes to pass the header of the message to the output.

Figure 17 investigates the effect of varied switch delay on the stall time performance of different memory management policies and switch design alternatives shown in the x-axis. In Fig 17, the switch size is held constant at 4 since we found that it performs better than a switch size of 2. Figures 18 and 19 also consider 4×4 switches. The results are plotted for three different memory management policies with both SWH and BVC switch designs. We notice that switch delay plays an important role when the switch design is limited to SWH. With BVC32, on the other hand, only moderate increase in average message latency is observed. The direct impact of increase in switch delay is an increase in the latency. Indirectly however, when the network gets congested, eg. for SWH and buddy policy, a small delay for one message causes other messages contending for the same link to get blocked for a long time. Such a blocking increases the overall average message latency and has a significant effect on the stall time performance.

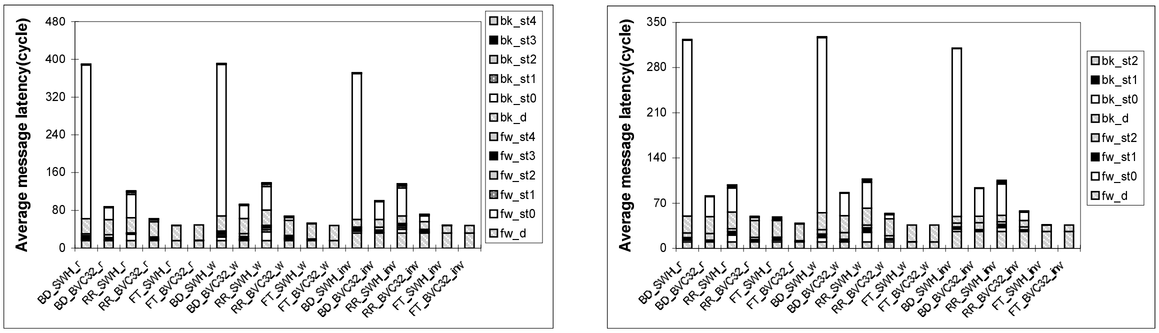
Figure 18 investigates the effect of varying the block size from 32 bytes to 64 bytes. This has two implications. First, it increases the length of data transfer for read and write transactions. Second, it may increase the coherence traffic for some applications due to false sharing within the larger cache block. The second implication is not so definite since the possibility of reducing the compulsory misses is quite high. If the application exhibits good spatial locality in its accesses to data, use of a 64 byte cache line will reduce the compulsory miss traffic by a factor of 2.

From Figure 18, we observe that the buddy policy and the round-robin policy suffer in average message latency with the SWH switch design when the block size is increased from 32 to 64. This effect is mainly due to the increase in the backward stage 0 (bk_st0) waiting delay. The amount of data being pushed out by the memory increases at the backward stage thus causing additional congestion at this stage. With first-touch, increasing the block size to 64 only increases the transmission time (fw_d). While considering the effect of block size, the average message latency cannot be used to convey its effect on the overall performance because doubling the block size effectively brings in twice the number of data items into the cache and can provide subsequent cache hits for an application with good spatial locality. This is reflected in the stall time results in Figure 18a. Improvements due to the increase in block size can be observed for all policies but in different stall time components. With the buddy memory management policy, the improvement is observed mainly in reduced latencies for memory service with invalidations (INVMS). By increasing the block size, the number of invalidations per block is reduced. The read and write network latencies do not reduce with an increase in block size in Figure 18a due to the increase in interference. On the other hand, while employing round-robin or first-touch, we observe a significant amount of improvement in read and write network latencies.



(a) Application Stall Time (b) Read Network Latency

Fig. 19. Effect of Varied Page Size



(a) Average Message Latency (2x2 switch) (b) Average Message Latency (4x4 switch)

Fig. 20. Effect on Different Transaction Types for Switch Sizes 2 & 4

Figure 19 investigates the effect of varying page sizes as 1Kb, 2Kb and 4Kb for the round-robin and first-touch page-based allocation policies. From the figure, we find that page size has little impact on the stall time and the average message latency for all memory management policies. However, we do observe a slight increase in the average message latency when the page size is increased from 1Kb to 2Kb when using round-robin policy. Then again it reduces as we shift from a page size of 2Kb to 4Kb. On the other hand, with first-touch, we see that there is no such effect of varying page size. The amount of data accessed by a processor is more than 4Kb, thus the access pattern effectively remains the same. The effect of page size for first-touch will be noticeable only if the amount of data accessed by a processor is lower than 4Kb or 2Kb. For example, if the processor accesses 2Kb data contiguously for a certain workload, runs with 1Kb size and 2Kb page sizes will behave identically. However the 4Kb size will result in a higher average message latency since it creates false sharing at the page level. Thus the effect of page size for first-touch depends highly on the application behavior. However, since many applications have a high data granularity, a page size of 4Kb seems acceptable.

In the previous few figures, we have used only the read network latency and shown its stage by stage delay within the network. However, different transactions could possibly exhibit different behavior in terms of waiting time at each switch. In particular, the invalidation request and a subsequent acknowledgement reply is much smaller than the read or write requests and replies. Figure 20 presents a comparison of read, write and invalidation message latency and their waiting delays in each stage of the network. The transactions are denoted by the suffix _r, _w and _inv respectively. Figure 20(a) uses a switch size of 2 with a switch delay of 2. Figure 20(b) uses a switch size of 4 with a switch delay of 2. We see that all transaction types follow the same behavior in average message latency. We notice similar improvements with improved switch designs for read, write as well as invalidation messages. It should be noted, however, that the FFT application usually requires invalidations for dirty copies in remote nodes. Thus each invalidation message has an update response that supplies an entire cache block to the memory. This is the reason for a huge delay at bk_st0 as in the case of reads and writes.

VI. Conclusion

In this paper we presented detailed simulation results for a CC-NUMA multiprocessor with different memory management policies and switch designs. Three different memory management policies, namely buddy, round-robin and first-touch, were implemented to study their impact on the network and memory system performance. We characterized the memory management policies based on their data placement technique and demonstrated that the temporal distribution of memory accesses of an applications differed greatly depending on the memory management policy. Five different scientific applications were considered and their memory access patterns were characterized.

Using the round-robin memory management policy, we evaluated four different switch architectures based on their switching, buffer length and virtual channels. Results for network traffic at various stages of a multistage network, and average message latencies and stall times of various applications were presented and analyzed. It was shown that there is a huge waiting delay at the network interface due to bulk requests to and replies from its memory module. Improvements in average message latency due to an increase in the number of virtual channels and buffer space were found to be as high as 70% for some applications. We measured the effect on read, write and invalidation messages separately and showed improvement in all cases. The effects of switch size and switch delay on the stall time performance of these applications were also presented.

We studied in detail the impact of memory management policies on the performance of interconnection networks. Results showed that the performance of buddy and round-robin memory management techniques are greatly improved when virtual channels are employed. The benefit is minimal for the first-touch policy. The impact of memory management on average message latency and stall time was measured and analyzed. We found that the degree of performance improvement depends largely on the memory management technique and switch architecture. Sensitivity of the results with respect to other parameters such as switch size, switch delay, block size and page size was minimal. Although we presented only FFT results with the variation of memory management, other applications behaved similarly.

References

[1] L.N. Bhuyan, Q. Yang and D.P. Agrawal, "Performance of Multiprocessor Interconnection Networks," *Computer*, vol. 22, no. 2, pp.25-37, Feb. 1989.

[2] L. N. Bhuyan, et al., "Performance of Multistage Bus Networks for a Distributed Shared Memory Multiprocessor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 1, pp. 82–95, Jan. 1997.

[3] 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, Cambridge, MA, September 1991.

[4] J. Carbonaro and F. Verhoorn, "Cavallino: The Teraflops Router and NIC," *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, Aug. 1996.

[5] 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.

[6] D. Chaiken, et al., "Directory-Based Cache Coherence in Large Scale Multiprocessors," *Computer*, vol. 23, no. 6, pp. 49-58, June 1990.

[7] M. Galles, "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip," *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, Aug. 1996.

[8] A. Kumar and L.N. Bhuyan, "Evaluating virtual channels for cache coherent shared memory multiprocessors," *ACM International Conference on Supercomputing*, Philadelphia, May 1996.

[9] R. P. Larowe and C. S. Ellis, "Experimental Comparisons of Memory Management Policies for NUMA Multiprocessors," *ACM Transactions on Computer Systems*, vol. 9, no. 4, pp. 319-363, Nov. 1991.

[10] D. Lenoski, et. al., "The DASH Prototype: Logic Overhead and Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 41–61, Jan 1993.

[11] D.J. Lilja, "Cache Coherence in Large Scale Shared Memory Multiprocessors," *ACM Computing Surveys*, vol. 25, no.3, pp. 303-338, Sept. 1993.

[12] 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.

[13] J. Ramanathan and L. M. Ni, "Critical Factors in NUMA memory management," In *Proc of the 11th Conference on Distributed Computing Systems*, pp. 500-507, Arlington , Texas, May 1991, IEEE Computer Society.

[14] C. Scheurich and M. Dubois, "Dynamic Page Migration in Multiprocessors with Distributed Global Memory," *IEEE Transactions on Computers*, vol. 38, no. 8, Aug. 1989.

[15] 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, pp. 5–44, March 1992.

[16] J. Torrellas and Z. Zheng, "The Performance of the Cedar Multistage Switching Network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 4, pp. 321–336, April 1997.

[17] A. Vaidya, A. Sivasubramaniam and C. Das, "Performance Benefits of Virtual Channels and Adaptive Routing: An Application-Driven Study," *Proc. 11th International Conference on Supercomputing*, Vienna, July 1997.

[18] B. Vergese, et al., "Operating System Support for improving data locality on CC-NUMA Computer Servers," *Proc. ASPLOS-VII*, pp. 279–289, Oct. 1996.

Laxmi Narayan Bhuyan

Laxmi Narayan Bhuyan (S'81-M'82-SM'87-F'98) received the M.Sc degree in Electrical Engineering from Sambalpur University, India, in 1979, and the Ph.D. degree in Computer engineering from Wayne State University, Detroit, MI, in 1982. At present, he is a professor of Computer Science at Texas A&M University, College Station, TX. His research interests are in the areas of computer architecture, parallel processing, interconnection networks and performance evaluation. He has published over 100 papers in these areas.

Dr. Bhuyan has served on the editorial boards of the IEEE Computer magazine, Journal of Parallel and Distributed Computing (JPDC), IEEE Transactions on Parallel and Distributed Systems and Parallel Computing journal. He was the Founding Program Committee Chairman of the First International Symposium on High-Performance Computer Architecture (HPCA), January 1995 and later Chairman of the IEEE Computer Society Technical Committee on Computer Architecture (TCCA) between 1996-1998. He is a Fellow of the IEEE.

Ravi R. Iyer

Ravishankar Iyer received his M.S. in Computer Science and B.S. in Electrical Engineering in August 1996 and December 1994 respectively from Texas A&M University, College Station, Texas. He is currently a Ph.D candidate in the Department of Computer Science at Texas A&M University. During his Ph.D, he has worked as a performance analyst at Intel Corporation and as a research intern at Hewlett-Packard Laboratories. His research interests include computer architecture, parallel computing and performance evaluation.

Hu-jun Wang

Hujun Wang received his MS degree in computer science from NanJing University of Science and Technology, China, in 1993, and BS degree in computer science from East China Shipbuilding Institute of Technology, China, in 1990. He is currently working toward his Ph.D degree in the Department of Computer Science at Texas A&M University. His current research interests include memory management, parallel applications, and interconnection networks.

Akhilesh Kumar

Akhilesh Kumar is a Staff Computer Architect in the Microprocessor Product Group at Intel Corporation since April 1996. He is a member of the IA64 Platform Architecture group, where he has made contributions in the areas of workload analysis, performance projection and processor bus and system interconnect definition. He finished his PhD in Computer Science at Texas A&M University in August 1996. He received his BE in Electronics Engineering from South Gujarat University, Surat, India, in 1988 and MTech in Computer Engineering from Indian Institute of Technology, Kharagpur, India in 1991. His research interests are in computer architecture and performance analysis.