ELSEVIER

Journal of Systems Architecture 48 (2002) 59–80

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Design and analysis of static memory management policies for CC-NUMA multiprocessors ☆

Ravishankar Iyer [a], Hujun Wang [a,b,*], Laxmi Narayan Bhuyan [a,c]

[a] *Intel Corporation, 15220 N.W. Greenbrier Parkway, Beaverton, OR 97006, USA*
[b] *Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA*
[c] *Department of Computer Science, University of California, Riverside, CA 92521-0304, USA*

Abstract

In this paper, we characterize the performance of three existing memory management techniques, namely, buddy, round-robin, and first-touch policies. With existing memory management schemes, we find several cases where requests from different processors arrive at the same memory simultaneously. To alleviate this problem, we present two improved memory management policies called skew-mapping and prime-mapping policies. By utilizing the properties of skewing and prime, the improved memory management designs considerably improve the application performance of cache coherent non-uniform memory access multiprocessors. We also re-evaluate the performance of a multistage interconnection network using these existing and improved memory management policies. Our results effectively present the performance benefits of different memory management techniques based on the sharing patterns of applications. Applications with a low degree of sharing benefit from the data locality provided by first-touch. However, several applications with significant sharing degrees as well as those with single processor initialization routines benefit highly from the intelligent distribution of data provided by skew-mapping and prime-mapping schemes. Improvements due to the new schemes are found to be as high as 35% in stall time.

Keywords: Memory management; Interconnection networks; Execution-based simulation; Scientific applications; Shared-memory multiprocessor

1. Introduction

Cache coherent non-uniform memory access (CC-NUMA) systems have become extremely popular since they are scalable and provide transparent access to data. With multiple levels of caches, they certainly provide cached data at low latencies. However, once the data access gets beyond the layers of cache, these machines pay a high penalty and their performance deteriorates. Cache misses are made up of local and remote memory accesses. Local memory access latencies are usually a magnitude higher than cache access latencies. The access time to a remote memory in a large system could be several orders of magnitude higher than the cache access time because of the

☆ This research has been supported by NSF grant MIP 9622740.
* Corresponding author.
E-mail address: hwang@cs.tamu.edu (H. Wang).

1383-7621/02/$ - see front matter
PII: S1383-7621(02)00066-8

time spent in the interconnection network. Even with low miss rates, the bottleneck in the performance of NUMA systems lies in the remote memory access latencies. Effectively, the cache miss latencies depend heavily on the ratio of local and remote accesses. The memory management policy governs the placement of data in shared memory. It specifies which memory accesses would be local and which would be remote. In this paper, our main aim is to present different memory management policies and study their impact on the application performance and interconnection network performance of a CC-NUMA multiprocessor system.

Related work in this area can be divided into two categories. The first category has been the performance evaluation of memory management policies [1–6]. Most of these studies [1–4] focused on distributed shared memory systems without hardware cache coherence. The effect of different policies was studied for CC-NUMA systems in [5,6]. Verghese et al. [5] presented significant data on the OS/hardware support required for dynamic page migration and replication policies. However, their results also indicate that dynamic memory management policies improve the performance of parallel applications (the SPLASH workload) by only 4% over the static schemes. Bhuyan, et al. [6] presented the impact of existing memory management policies and switch design alternatives on the application performance. Hence we concentrate only on static memory management techniques. We propose two improved static schemes: (1) skew-mapping, (2) prime-mapping that significantly improve the performance of several applications over the existing schemes. The skew-mapping scheme is based on skewing pages which are allocated to memories using the round-robin policy. It can be specified as a function that maps logical pages onto memories such that the required pages could be accessed conflict-free. Several general classes of skewing memory data accesses have been investigated and characterized by previous researchers [7–10], but not yet been developed for CC-NUMA memory management. The prime-mapping scheme is based on the allocation of data pages to memories according to a prime number. The use of a prime number for effective distribution of data accesses has been studied in a few papers [11–13]. Lawrie [11] described a memory system designed for parallel array access which is based on the use of a prime number of memories in SIMD computers. Yang [12] presented a prime-mapped cache in the vector processing environment. The memory access logic of improved allocation schemes is similar to those of the existing policies, resulting in no additional delay for memory accesses. Further, the prime-mapped cache for vector computers was evaluated using a set of applications in [13]. However, they have not yet been developed for CC-NUMA multiprocessors.

The second category of related work is the performance evaluation of interconnection networks (IN's). Performance evaluation of IN's has been an active area of research for a long time [14–18]. These studies were conducted with a synthetic workload that is more suitable for a message passing or networking environment. Workload in a CC-NUMA environment is characterized by unsymmetrical bulky messages due to cache coherence, synchronization and memory management. Furthermore, due to the IN advancements, such as the use of virtual channels (VC's), the improvements in system performance have to be judged from the dynamic changes during the execution of applications. The aim of this paper is to re-evaluate the performance of an IN with realistic application data accesses and memory management policies governing the placement of the data blocks. Execution-based evaluations for IN's have been reported [19–21] to test the effectiveness of VC's. While these studies provide useful data, they do not explore different memory management techniques as we do. We consider a multistage interconnection network (MIN) in this paper, similar to the one employed in Butterfly and Cedar multiprocessors [22]. Unlike Cedar, we employ a NUMA organization with one network (like Butterfly) that is used for both forward and backward (reply) messages. We evaluate two different switch design alternatives (*simple wormhole* (*SWH*) *and buffered virtual channel* (*BVC*)) [20] for the MIN.

To evaluate the different memory management techniques in conjunction with different switch architectures, we have significantly modified our CC-NUMA simulator based on Proteus [23]. We

developed virtual memory support in the Proteus simulator to evaluate the memory management techniques. We have also incorporated detailed switch models and wormhole routing with VC's in the network to accurately interpret the effect of network latency on application stall time. We have modified the directory based cache coherence protocol [24] in the simulator to correctly account for the possible out-of-order messages due to the presence of VC's in the MIN. Our simulation results show that the performance of memory management techniques depends on the application sharing pattern. Applications with a low degree of sharing benefit from memory management techniques such as first-touch. However applications with initialization routines or moderate to high sharing degrees do not enjoy the same benefits. For several such applications, skew-mapping and prime-mapping schemes provide significant benefits. The paper makes the following contributions:

- We study and evaluate a spectrum of current static memory management schemes for CC-NUMA multiprocessors. As a result, we present insights regarding the bottlenecks that limit application performance.
- We develop two improved static memory management policies, called skew-mapping and prime-mapping, to improve the performance of CC-NUMA systems.
- We characterize the effect of page placement on the temporal and spatial locality of memory access patterns for several scientific applications.
- In order to investigate the performance impact of existing and improved memory management policies accurately and extensively, we employ an execution-driven simulation methodology that fully models the interconnection network design with several choices for crossbar switch design.
- Finally, we analyze application performance for these memory management policies at the system level and the network level. Interference at different stages in the MIN is also captured and analyzed.

The rest of the paper is organized as follows. Section 2 gives an overview of the existing memory management policies and the details of two improved memory management policies. Section 3 describes the experimental methodology. Section 4 presents not only the data-sharing patterns of each application, but also temporal and spatial locality characteristics of each application which are correlated with each memory management policy. Section 4 also presents a detailed study of memory access patterns for a single application, namely fast Fourier transform (FFT), to accurately depict the effect of the memory management policies on the execution flow of the application. Section 5 presents the impact of the policies and the effect of the switch architectures on the network latency and application stall time. Finally, Section 6 presents the conclusions.

2. Memory management policies

Memory management policies in CC-NUMA architectures should be targeted to address the disparity between access times for data located in caches, local memories and remote memories. A typical CC-NUMA system architecture is shown in Fig. 1. Fig. 1 also conceptually points out the increase in data access latency as the data item gets accessed from the L1 cache, the L2 cache, the local memory and the remote memory.

Since caches cannot hold all the data due to space and coherence restrictions, ideally we would like the local memory to handle all cache misses.

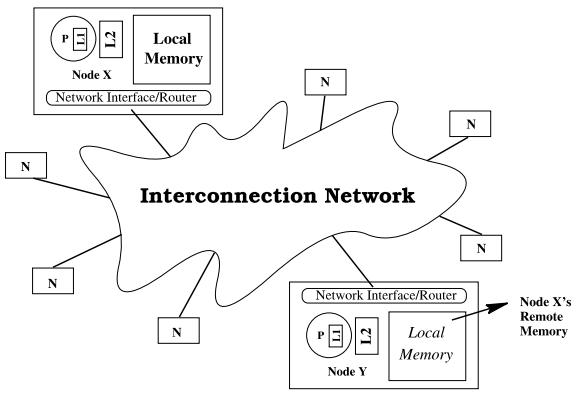

Fig. 1. Conceptual view of a CC-NUMA system.

Unless there is absolutely no data sharing in the application, it is impossible to statically allocate data in a manner in which all nodes see the required data in their respective local memories. Finally if data is remotely allocated, we should be careful to provide the access in such a way as to distribute the load and reduce network interference and memory congestion. The size of the data chunk allocated may also play an important role in the performance of a given memory allocation scheme since it dictates the positioning of several data items and their access patterns.

2.1. The existing policies

We consider the following memory management policies to address the above issues.

Buddy allocation: This scheme is invariably used for uniprocessors and SMP servers. 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 two. 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. However, as we shall see, the performance is poor in a CC-NUMA environment since it saturates a particular memory with multiple accesses from other processors. Data accesses are directed towards a selected few nodes in the system, causing severe memory and/or 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 N, where N 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 accesses to different data from different processors are interleaved among the different nodes. This scheme is used in the DASH [25], HP/Exemplar [26], and SGI Origin [27] 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. This queues up requests at the network interface and the memory subsystem.

First-touch allocation: In this allocation scheme, each page is located at the processor that first accessed the page. In other words, at the time of the first page fault, it is allocated to the local memory of the faulting processor. The advantage to this scheme is that each page is positioned with limited knowledge of initial application behavior. A well-written application can exploit the use of this scheme and certainly gain in performance. On the other hand, this scheme notices only the first access to the page. Poorly written applications (eg: initializations performed at a single processor before main parallel computation begins) will locate pages based incorrectly on the first access and cause remote memory accesses during later accesses to the page by other processors. The first-touch allocation scheme is the default policy which is provided in the SGI origin system (The round-robin policy is a choice for the user).

2.2. The improved memory management policies

For the round-robin policy, we find that requests from different processors go to the same memory at the same time giving rise to bulk of replies (in the form of data blocks) from the same memory [20]. The cause of this bulky response lies in the distribution of scientific data of size of a power of two distributed over a number of memory modules, also a power of two. This phenomenon builds up a large delay at the network interface and considerably affects the performance. In this section, we introduce two improved memory management policies which improve performance by allocating data to memory modules in a more intelligent fashion. The basic idea behind these schemes is to allocate pages in a non-regular manner so that applications with regular access patterns do not generate multiple accesses directed to the same memory module simultaneously.

2.2.1. Skew-mapping policy

Definition 1. A page-allocation scheme for M pages and N processors is a function

$$S : \{0, 1, 2, \ldots, M-1\} \rightarrow \{0, 1, \ldots, N-1\} \quad (1)$$

where $S(i) = j$ means page i is allocated to processor j.

Definition 2. A round-robin scheme for M pages and N processors can be expressed as the following function:

$$S(i) = i \bmod N, \quad (0 \leqslant i \leqslant M-1) \quad (2)$$

A skewing scheme is a function which maps logical pages into processors such that, hopefully, the required pages could be accessed conflict-free.

Definition 3. Given M pages and N processors, a skewing scheme is a function

$$S(i) = \left(i + \left\lfloor \frac{i}{N} \right\rfloor + 1\right) \bmod N, \quad (0 \leqslant i \leqslant M-1) \quad (3)$$

where $S(i)$ skews M pages linearly in the *left-to-right* direction and $\lfloor \frac{i}{N} \rfloor$ denotes the greatest integer less than or equal to i/N. It is important to see that the mapping could be done at the compile time, or it can be done at the hardware level by using a shifter.

An example of 32 pages and 4 processors is shown in Fig. 2. Fig. 2(a) illustrates that the pages are allocated to processors using the round-robin policy. That is, the first 4 pages are allocated to processor P_0, P_1, P_2, and P_3, the next 4 pages are allocated to processors P_0, P_1, P_2, and $P_3, \ldots$, and so on. Fig. 2(b) shows that the pages are allocated to processors using the skew-mapping policy. Note that the skew-mapping policy has the effect of skipping one processor for every 4 pages allocated. That is, the first 4 pages are allocated to processors P_1, P_2, P_3, and P_0, the next 4 pages are allocated to processors P_2, P_3, P_0, and $P_1, \ldots$, and so on.

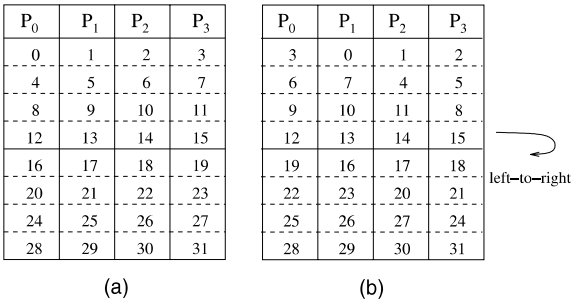

Fig. 2. An example of skew-mapping policy: (a) round-robin, (b) skew-mapping scheme.

2.2.2. Prime-mapping policy

Definition 4. Given M pages, N processors, and a prime number $P(N \leqslant P)$, a prime-mapping scheme can be divided into three steps:

- The M pages are allocated to P processors using equation $S(i) = i \bmod P$ (i is a page number), where processors $P_N, P_{N+1}, \ldots$, and P_{P-1} are virtual. That is, the first P pages are allocated to processors $P_0, P_1, \ldots$, and P_{P-1}, the next P pages are allocated to processors $P_0, P_1, \ldots$, and $P_{P-1}, \ldots$, and so on.
- We reorder those pages, which are allocated to virtual processors, as pages $0, 1, \ldots, j$. These reordered pages are allocated to processors $P_0, P_1, \ldots$, and P_{P-1} using equation $S(i) = i \bmod P$ again.
- Repeat step 2 till these M pages are allocated to processors $P_0, P_1, \ldots$, and P_{N-1}

An example of 20 pages and 4 processors with *prime* number 5 is shown in Fig. 3. Fig. 3(a) illustrates that the 20 pages are allocated to 5 processors using the round-robin policy (i mod 5),

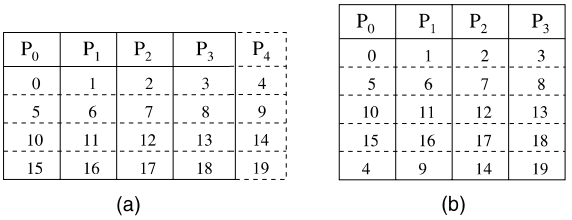

Fig. 3. An example of prime-mapping policy: (a) round-robin to five virtual processors, (b) actual prime-mapping.

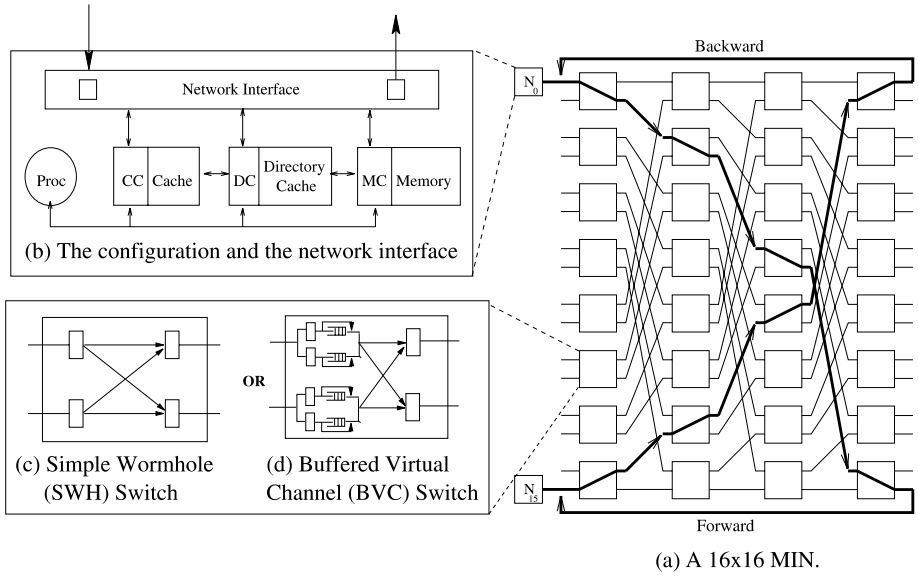

Fig. 4. Overview of the system architecture.

where processor 4 is virtual. That is, the first 5 pages are allocated to processors P_0, P_1, P_2, P_3, and P_4, the next 5 pages are allocated to processors P_0, P_1, P_2, P_3, and $P_4, \ldots$, and so on. Fig. 3(b) shows how to allocate the pages, which are allocated to processor 4, to other processors. Note that we reorder pages 4, 9, 14, and 19 as pages 0, 1, 2, and 3, respectively. These reordered pages are allocated to processors P_0, P_1, P_2, and P_3 using i mod 5 again.

3. Experimental methodology

Our simulator is based on PROTEUS [23] that implemented MINs using an analytical model. We have modified the simulator extensively to exactly model the bi-directional MIN with wormhole routing. We have also incorporated different switch architectures with VC's and multiflit buffers [6,20]. For this work, all the virtual memory support needed to simulate the data allocation policies in Section 2 has been implemented. We implemented the full-map directory-based cache coherence protocol [24] with some modifications for evaluation, as described in [6]. The scheme is similar to the scheme used in the MIT Alewife system. The architecture under consideration can be modeled as processor, cache, memory, network interface and the network, as shown in Fig. 4(b). We choose a 16-node system to limit the simulation time, a larger system would need an appropriately larger data structure to get realistic results. The cache parameters used in the simulation are similar to the HP PA-8000. The system parameters used in the simulation are listed in Table 1.

Table 1
Simulation parameters

Parameter	Value
Number of processors	16
Shared memory size per node	512Kbytes
Cache size	32Kbytes
Cache line size	32 bytes
Number of sets	2
Cache access time	2 cycles
Memory access time	50 cycles
Switch delay	4 cycles
Switch size	2×2 or 4×4
Link width	16 bits
Flit length	16 bits
Message lengths	8 or 40 bytes
Page size	1Kbytes
Interface delay	20 cycles

3.1. Network/switch architecture

The schematic of the network is shown in Fig. 4(a). It is a MIN employing 2×2 switches. The interconnection between them is a perfect shuffle. We use wormhole routing and virtual cut-through switching techniques because they give better performance than packet switching and because they are used in the design of current multiprocessor switches [28,29]. 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 of 16 bits each long. The *rddata* and *wrdata* messages supply the block to/from the requesting cache and are 20 flits long for a cache line size of 32 bytes.

A SWH 2×2 crossbar switch is shown in Fig. 4(c). The amount of buffer needed per input and output for wormhole routing is only one flit that is absolutely essential for transmission over the links. The SWH switch does not have any virtual channel. Fig. 4(d) shows a BVC switch [6,20]. The BVC32 used in this study has a buffer size of 32 flits and 2 VC's. VC's [15] are used in wormhole networks to avoid deadlocks and to improve link utilization and network throughput. The switch delay is modeled to be 4 cycles, which is similar to the time taken in SGI Spider and Intel Cavallino switches [28,29]. The SWH and BVC crossbar switches were described in detail [6].

3.2. Benchmark applications

We have selected five numerical applications as the workload for evaluating the cache-coherent shared-memory multiprocessor. These applications are: (1) multiplication of two 2-D matrices (MATMUL) which was done between two 128×128 double precision matrices; (2) Floyd–Warshall's all-pair-shortest-path algorithm (FWA) which used a graph of 256 nodes with random weights assigned to the edges; (3) blocked LU factorization of a dense 2-D matrix (LU) which was done on a 256×256 matrix using 8×8 blocks; (4) 1-D FFT where the simulations are done on an input of 2^{14} points; (5) simulation of rarefied flows over objects in a wind tunnel (MP3D) where we used 25,000 molecules with the default geometry provided with SPLASH [30] and the simulation was done for five time steps. These applications were described in detail [6].

4. Application-centric results

We present data-sharing patterns, local and remote access probabilities, interarrival rates, and memory access patterns to compare the varied impact of memory management policies on the workload data access behavior.

4.1. Data sharing pattern of applications

The emphasis here is on the pattern of sharing that is inherent in applications themselves, rather than that caused by the underlying memory system architecture, or the cache coherency protocol. Our characterization of sharing serves two purposes: (1) it provides an understanding of the memory reference patterns of data sharing; (2) it explains how different patterns of sharing can affect system performance, such as stall time, etc. Data sharing can be divided into read sharing and write sharing. Read(write) sharing corresponds to memory references that cause sharing misses of type read-(write). A classification of read or write shared data can be done based on the granularity of the data block:

- *Degree of read sharing* at page level is the number of processors which read a page.
- *Degree of write sharing* at page level is the number of processors which write a page.

Figs. 5–9 show the read and write sharing patterns, and list the degree of read and write sharing in the left and right graph of each figure, respectively. The *y*-axis represents the number of pages that are shared by the number of processors on the *x*-axis. Now, we examine these graphs:

FFT Application: Our simulation experiments used a 2^{14}-pt FFT with 16 processors. Thus the application run should consist of ten *local* initial stages and four *read-after-write* (RAW) sharing stages. In these initial stages, processors only read and write their local data points. After these initial

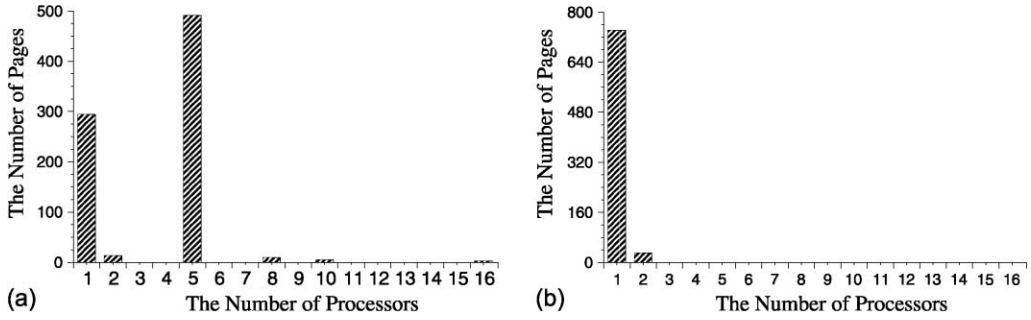

Fig. 5. Data sharing pattern for FFT application: (a) read sharing at page level, (b) write sharing at page level.

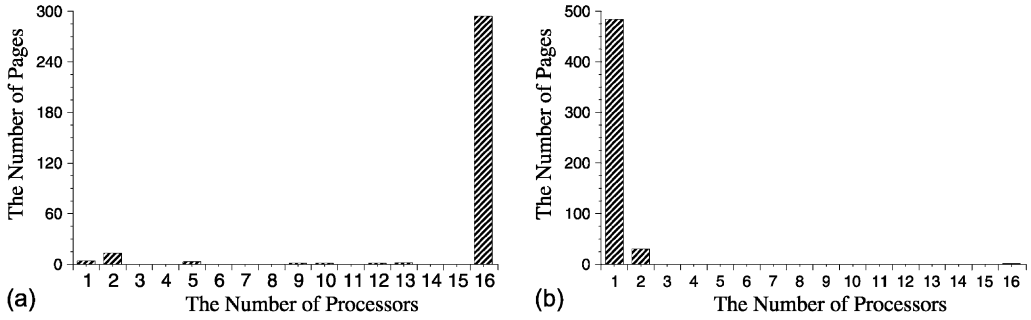

Fig. 6. Data sharing pattern for FWA application: (a) read sharing at page level, (b) write sharing at page level.

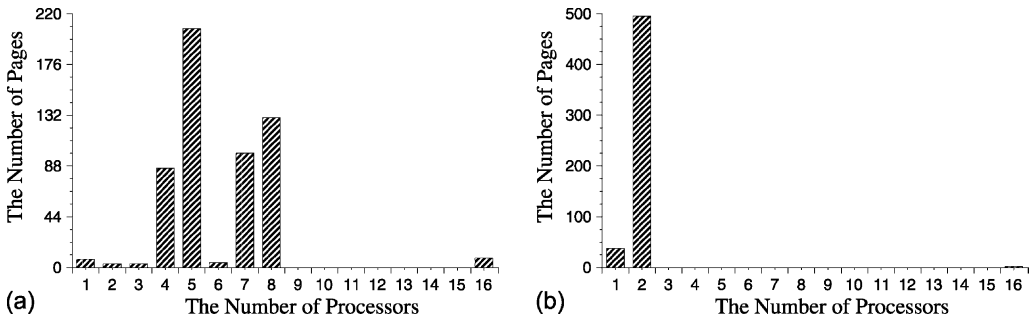

Fig. 7. Data sharing pattern for LU application: (a) read sharing at page level, (b) write sharing at page level.

stages are completed, these processors exchange the data produced at the end of each RAW sharing stage. The data point is read by a different processor in each sharing stage. The data point, however, is written only by one processor in each sharing stage. Because of four RAW sharing stages, the data point is read by four processors. In Fig. 5(a), there are 298 and 491 pages whose degree of read sharing is equal to 1 and 5, respectively. In Fig. 5(b), there are 742 pages whose degree of write sharing is equal to 1.

FWA Application: We used 256 nodes and 16 processors in the experiment. The shared 2-D arrays consist of one distance matrix and another predecessor matrix. The problem is partitioned as per the rows in the distance matrix, so a set of

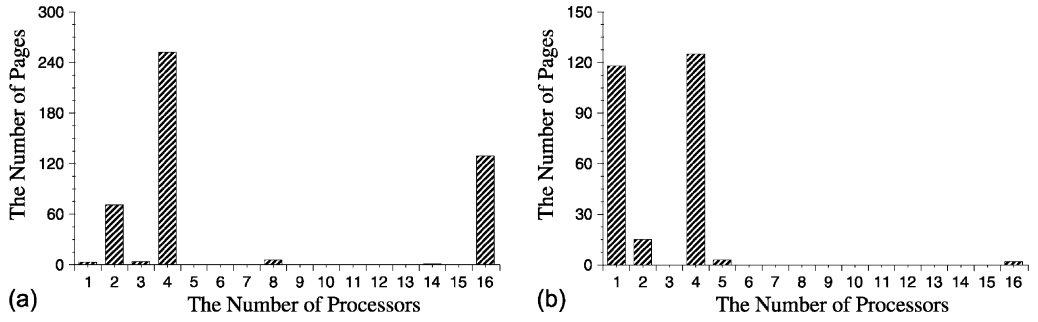

Fig. 8. Data sharing pattern for MATMUL application: (a) read sharing at page level, (b) write sharing at page level.

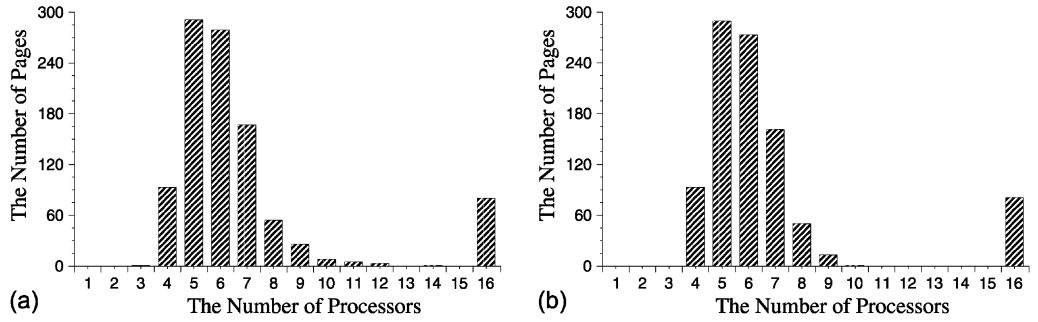

Fig. 9. Data sharing pattern for MP3D application: (a) read sharing at page level, (b) write sharing at page level.

vertices is statically allocated to a processor to compute the shortest paths from them. Therefore, the elements in the distance and predecessor matrices are written by only one processor during the whole execution. A particular row of distance and predecessor matrix, however, is read by 16 processors. In Fig. 6(a), there are 294 pages whose degree of read sharing is equal to 16. In Fig. 6(b), there are 484 pages whose degree of write sharing is equal to 1.

LU Application: This experiment contains 2-D arrays in which the first dimension is the block to be operated on, and the second contains all data points in that block. There are four rows, each of which has four processors, and four columns, each of which has four processors. In this manner, all data points in a block (which are operated on by the same processor) are allocated contiguously and locally. For iteration $k(1 \leqslant k \leqslant 4)$, there are four steps: (1) factorize a diagonal block (k,k); (2) divide column k by the diagonal block; (3) modify row k by the diagonal block; (4) modify subsequent block columns. Thus, most of data points are read by 4 or 5 processors, and are modified by two processors as shown in Fig. 7.

MATMUL Application: Two 128×128 data matrices (A and B) and 16 processors are used here. Parallel matrix multiplication does $A[128 \times 128] \times B[128 \times 128] = C[128 \times 128]$. The result matrix C is divided among processors and each processor calculates its portion of matrix C. C is divided row and column wise such that the number of rows and columns of C computed by a processor is almost of same size, i.e., matrix C is equally divided into 16 portions. So, each data point is read by four processors and is written by one processor. In Fig. 8(a), there are 252 pages whose degree of read sharing is equal to 4. In Fig. 8(b), there are 118 pages whose degree of write sharing is equal to 1, and 125 pages whose degree of write sharing is equal to 4 because there are many data points at the boundaries of partitions.

MP3D Application: MP3D employs a five degree-of-freedom simulation of idealized diatomic molecules in a 3-D active space. The work is partitioned by the total of 25,000 molecules, which are statically scheduled on processors. Molecules generally flow through the tunnel in the positive x direction. The active space is represented as a 3-D space array of unit-sized cells. Data sharing occurs during collisions and during accesses to the space array. Molecules can move among cells, but are only eligible for collision with other molecules occupying the same cell at that time. Molecular collisions are statistically determined using a collision probability, conservation laws, and a table of collision outcomes. In Fig. 9(a), there are 986 pages whose degree of read sharing is more than or equal to 4. In Fig. 9(b), there are 971 pages whose degree of write sharing is more than or equal to 4.

Above, we presented the data sharing patterns of several scientific applications. While this data certainly helps understand the characteristics of the applications, it also directly relates to the choice and performance of static memory management policies. When the data sharing degree of applications is low, a memory management policy such as first-touch might be appropriate since it provides local memory access. On the other hand, when the sharing degree is high, several processors access a page during the execution of the application. Thus the initial placement of pages based on first-touch does not suffice. In such scenarios, the problem translates to providing simultaneous access to remote data in an efficient manner, so as to provide fast remote access without hot-spots. We find that most of the applications have significant sharing degrees. Thus, these applications should significantly benefit from memory management policies such as skew-mapping and prime-mapping that are aimed at providing efficient remote access by distributing data in an intelligent manner.

4.2. Effect of memory management policies

We begin to investigate the behavior of the memory management policies by observing how

Table 2
Variations in memory access probabilities

Application	PID	Buddy (%)				Round-robin (%)				First-touch (%)				Skew-mapping (%)				Prime-mapping (%)			
		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
FWA	0	0	0	0	100	24	22	24	30	95	2	2	1	20	23	30	27	43	17	18	22
	1	0	0	0	100	23	23	26	27	3	94	2	1	22	24	26	28	28	27	22	23
	2	0	0	0	100	24	25	24	27	3	2	93	1	22	21	31	26	29	23	24	23
	3	0	0	0	100	26	23	23	29	3	2	2	93	22	21	24	33	24	20	27	29
MAT-	0	0	0	0	100	23	24	26	27	53	4	43	0	25	26	24	25	26	26	24	23
MUL	1	0	0	0	100	25	22	26	28	10	47	0	43	25	25	25	25	28	22	24	26
	2	0	0	0	100	27	26	24	24	46	0	51	3	25	25	25	25	27	26	24	23
	3	0	0	0	100	28	28	24	21	3	43	7	47	25	25	25	25	28	23	23	26
LU	0	0	0	0	100	30	23	24	23	97	2	1	0	25	24	25	26	25	24	25	26
	1	0	0	0	100	31	21	24	23	99	1	1	0	25	22	24	29	28	25	23	24
	2	0	0	0	100	30	22	24	24	98	2	1	0	23	21	27	29	26	24	24	25
	3	0	0	0	100	30	22	24	24	99	0	1	0	24	25	24	27	25	25	25	25
MP3D	0	1	0	40	59	26	25	25	24	23	9	17	52	25	25	26	24	24	21	24	31
	1	1	0	40	59	26	25	25	25	20	10	18	52	25	25	26	24	24	21	24	31
	2	1	0	40	59	26	25	25	25	19	9	20	52	24	25	27	24	23	21	24	32
	3	0	0	41	59	25	25	25	25	18	9	18	55	25	26	26	23	23	21	24	32
FFT	0	0	0	0	100	26	26	24	24	88	7	5	0	25	25	25	25	26	24	25	25
	1	0	0	0	100	26	26	24	24	6	89	0	5	26	25	25	24	24	25	26	25
	2	0	0	0	100	26	26	23	24	7	0	85	8	25	25	25	25	27	25	24	24
	3	0	0	0	100	26	26	24	24	0	5	6	89	25	25	25	25	25	24	26	25

the memory requests are distributed in CC-NUMA architectures for each application. Table 2 shows the percentage distribution of the data accesses for each of the workloads under different memory management policies. The id of the processor (PID) generating the memory accesses is listed in the second column and the memory modules to which the requests are addressed are presented in the following columns. We observed similar results for the 16-node system but chose to show the 4-node results for brevity. For FWA, MATMUL, LU and FFT applications, it is clear that the whole chunk of shared data gets allocated in a single node when the buddy memory management policy is used, so all processors access memory module 3 all the time. We also notice that the shared data for the MP3D application get allocated in two nodes. This is due to the memory size of 512Kbytes and the data size between 512Kbytes and 1024Kbytes. 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% probability. With first-touch we find that this local memory access probability increases tremendously for FWA and FFT applications. Thus, the number of remote requests and responses that traverse the network is reduced tremendously. This leads to a considerable improvement in the average stall-time performance. We find that round-robin, skew-mapping and prime-mapping equally distribute the accesses over different memories in the system. However, we will observe that the skew-mapping and prime-mapping schemes improve the performance over round-robin due to a better temporal distribution of processor accesses.

Having observed the spatial distribution of memory accesses, we now concentrate on the temporal locality of a processor's request. The *think time* of a processor is the time between when a cache miss is satisfied and the next one is generated. The *interarrival time* is defined as the time between two cache misses that includes the memory and network access time. Figs. 10–14 present the cumulative distribution function (CDF) of the interarrival time for different memory management policies. Each figure consists of six curves; one representing the workload think time and five others representing the interarrival times for each of the memory management policies. Fig. 10 clearly shows the difference between latencies

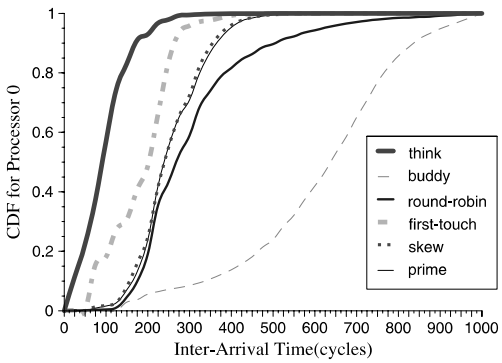

Fig. 10. Interarrival time distribution for the FFT application.

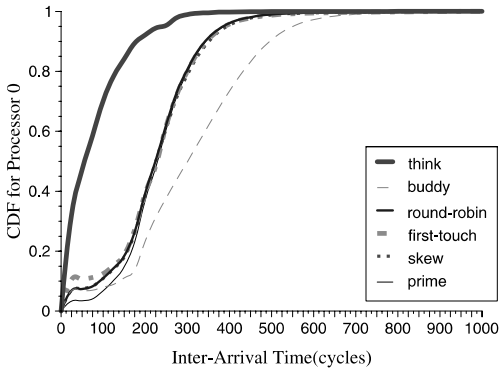

Fig. 11. Interarrival time distribution for the MP3D application.

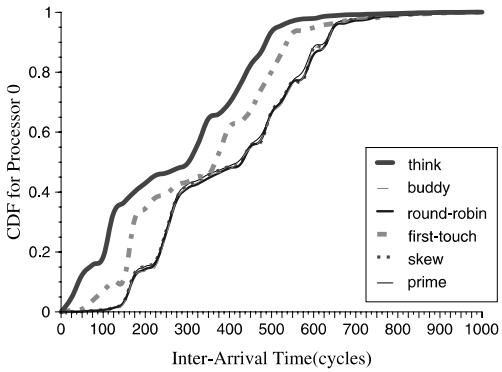

Fig. 12. Interarrival time distribution for the LU application.

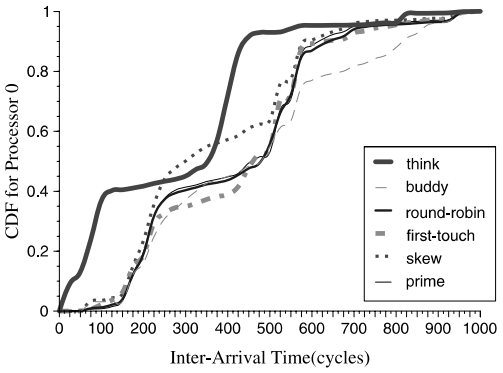

Fig. 13. Interarrival time distribution for the MATMUL application.

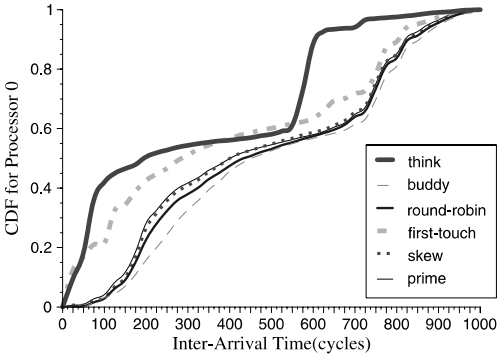

Fig. 14. Interarrival time distribution for the FWA application.

observed for each memory management policy for the FFT workload, where the think-time curve is on the extreme left. The lowest latency for memory access is provided by first-touch with an interarrival time lower than 275 processor cycles during 93% of the parallel execution time. The skew-mapping and prime-mapping policies provide latencies of 375 processor cycles. On the other hand, round-robin provides a latency of 575 processor cycles and buddy provides a latency of approximately 875 processor cycles for the same period of time. Compared to the buddy memory management policy, the first-touch policy improves the average memory access latency by approximately 69%, skew-mapping and prime-mapping policies further improve the average memory access latency by approximately 57%, and round-robin further improves by 34%.

From Figs. 10–12, we find that temporal access patterns for FFT, MP3D and LU applications fit CDFs of known distributions like exponential or hyper-exponential distribution. Similar observations were made [31]. However, FWA and MATMUL applications exhibit jumps in the interarrival times. Fig. 13 presents a probability of approximately 40% that the think time is less than 125 processor cycles and a probability of close to 50% of a think time higher than 375 processor cycles for the MATMUL application. 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 ($\leqslant$125 processor cycles). Once these cold misses are served and the rows/columns are read into the caches, subsequent computation that requires these elements are cache hits. From then on, cache misses become infrequent and occur only while performing writes to elements of the result matrix. Since these misses are in the outer loop of the program with dot-product computation of a row and column of the input matrices within, the think time is high ($\geqslant$375 processor cycles). It is difficult to characterize the CDF of FWA and MATMUL applications by known distributions. In Figs. 13 and 14, we also notice that the memory management curve crosses over the think time to give an impression that the think time is higher than the overall interarrival time for a realistic execution. However, this is not the case. Since the curve represents a CDF of interarrival time, this should be interpreted as percentages of requests that have a certain value for interarrival time. Since the think time distribution progresses in an irregular manner, it is difficult to identify exact reasons for such an occurrence.

4.3. The FFT access pattern

Representing application behavior is not an easy task since each application runs for millions of processor cycles and there are several thousands of requests. However we attempt here to present the characteristics of a single FFT application in order to conceptualize the data access behavior

during each parallel segment of the application. Our aim here is to present the changes in the application data access patterns to local and/or remote memories by varying the memory management policy.

An M-point FFT application consists of $\log_2 N$ stages of butterfly computation. Given N processing elements for parallel computation, the bit-reversed input data array is equally divided among the processing elements. Thus, each processor is responsible for the computation of M/N elements of data. An example of a 16-pt FFT computation using four processing elements is shown in Fig. 15. Since the computation follows a butterfly pattern, the application can be divided into two different types of stages. Fig. 15 shows that the first two ($\log_2 (M/N)$) stages are performed *locally* on disjoint sets of data. Here locally means that the data points accessed by one processor are not touched by any other processor during these stages. We refer to these stages as the *local computation stages*. Once the local stages are complete, the inherent communication of the application begins. The last two ($\log_2 N$) stages of computation require data exchange among the processors. Each arrow within a stage represents the data exchange, with a read (R) at its tail and a write (W) at its head. This causes RAW type sharing conflicts between consecutive stages. A barrier synchronization is used between the stages to ensure the correct ordering of the conflicting accesses. These stages are referred to as the RAW computation stages. Our simulation experiments used a 2^{14}-pt FFT with 16 processors. Thus the application run should consist of ten ($\log_2(2^{14}/16)$) *local* stages and four($\log_2 16$) RAW stages. The application uses two data structures (*local array* and *RAW array*) based on the names of the stages in which they are accessed. Two arrays are used to avoid a large number of conflict misses.

Fig. 16 shows the FFT memory access pattern to memory 15 when employing the buddy memory management policy. The contiguous allocation of data in the buddy policy causes the local array to get stored at a single memory, while the RAW array is stored at a single memory (15). All other memory modules do not hold any shared data and are not accessed for read/write transactions by any processor. In Fig. 16, the x-axis represents the parallel execution time, while the y-axis represents the ID of the processor that generated the read/write transaction. Fig. 16 plots the accesses during the RAW stages from all processors to the RAW data array in memory module 15. Note that there is a higher number of memory accesses during these four RAW stages when compared to the few

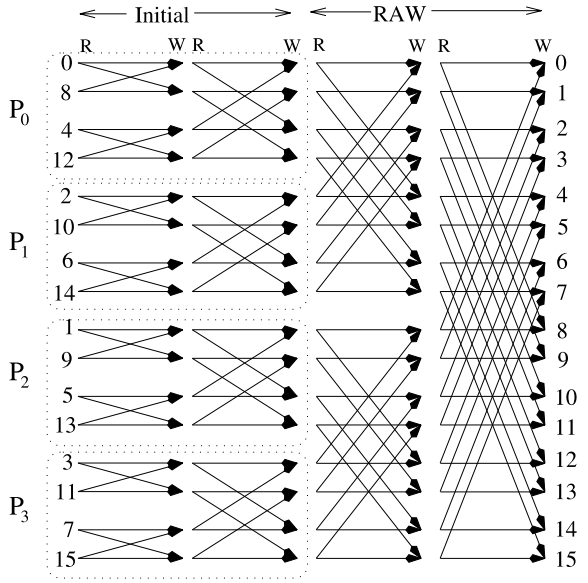

Fig. 15. A 16-pt FFT data flow graph using four nodes.

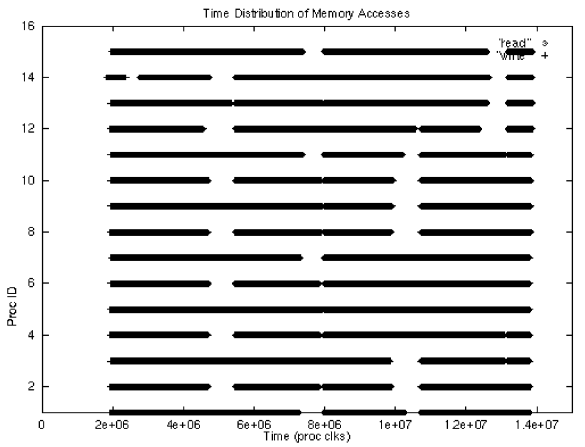

Fig. 16. FFT memory access patterns to memory module 15 using the buddy policy.

cache misses during the local stages. Since a single memory serves all processors' requests, the time taken for each RAW stage to complete is as high as 3 Mcycles; almost an order of magnitude higher than a local stage's average execution time. This is due to the memory hot-spot created by the buddy allocation policy. Finally we note that the entire computation ends with a reorganization of data and a parallel execution time of approximately 14 Mcycles. We will use these execution times as a base for evaluating the performance benefits of round-robin and first-touch memory management policies.

Fig. 17 shows the data access patterns using the round-robin policy. Unlike the buddy policy, the round-robin scheme allocates the two data arrays on a page-by-page basis distributing it evenly across all the memory modules in the system. Thus when compared to the buddy allocation scheme, this scheme has much fewer data accesses to a single memory module. Fig. 17 represents the application access patterns to memory module 15. All other memory modules also exhibit similar access patterns. Though round-robin distributes the pages evenly across all memory modules, the distribution is very regular because a data size of a power of 2 is serially distributed over a number of memory modules which is also a power of 2. Thus we find that many processors generate simultaneous read/write accesses to different pages on the same memory module. Examples of several such simultaneous accesses starting at time 1.9 Mcycles at intervals of approximately 1.5 Mcycles can be found in Fig. 17. The first ten local stages complete within the same time as the buddy allocation policy since the number of cache hits is high and memory accesses are limited. However with a uniform access distribution, much of the hot-spots at the memory is alleviated during the last four RAW stages. The per-stage delay for RAW stages is reduced to 1.5 Mcycles (when compared to 3 Mcycles for buddy), an improvement of over 50%. Note also that the overall parallel execution time reduces to 8.4 Mcycles; an improvement of approximately 40%.

Fig. 18 shows the memory access patterns for the first-touch policy. We see that the FFT application characteristic is well utilized by this policy. As expected, we find that each memory is accessed more frequently only by its own local processor when the first-touch policy is employed. Since the initial (local) FFT stages define the application data access behavior, the first touch policy succeeds in using the first page fault as a means of providing a high degree of data locality. As we saw earlier in Table 2, almost 90% of the data accesses are to the local memory when the first-touch scheme is employed. From Fig. 18, one can notice that the first RAW stage consists of read data accesses to memory module M_{15} from a node

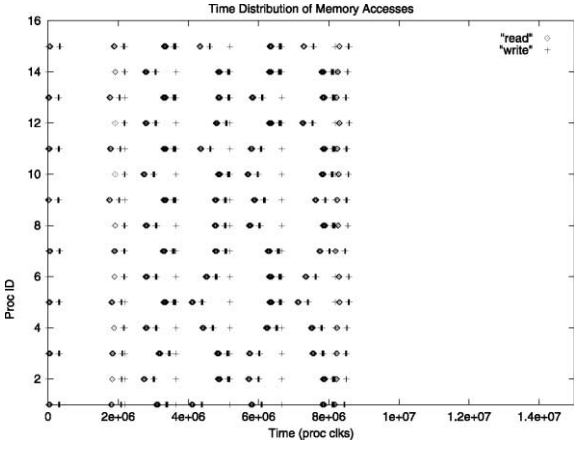

Fig. 17. FFT memory access patterns to memory module 15 using the round-robin policy.

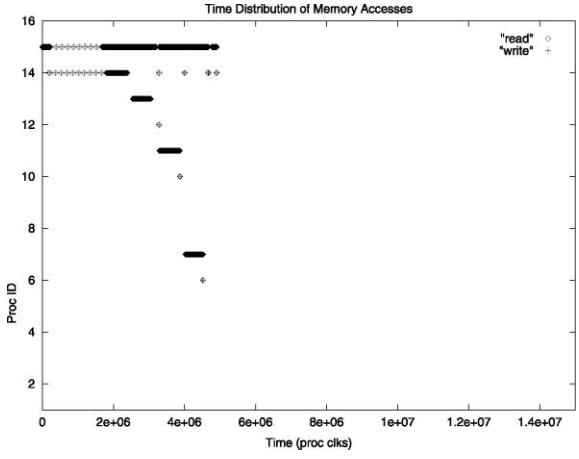

Fig. 18. FFT memory access patterns to memory module 15 using the first-touch policy.

1-away (P_{14}), while the next RAW stage accesses data from the node that is 2-away (P_{13}) from it. RAW stages 3 and 4 consist of accesses from 4-away (P_{11}) and 8-away nodes (P_7) as expected. All other memory modules (M_0 to M_{14}) also exhibit similar patterns. By utilizing the application behavior and placing data intelligently, the first-touch scheme is able to reduce the RAW stage delay to 0.75 Mcycles. The overall parallel execution time is reduced to 4.4 Mcycles (when compared to 14 Mcycles for the buddy policy): an improvement of approximately 69%. The improvement is mainly due to savings while executing the last four stages of the butterfly computation.

Figs. 19 and 20 show the data access patterns using the skew-mapping and prime-mapping policies, respectively. By utilizing the properties skewing and prime, the new memory management designs improve the performance of CC-NUMA multiprocessors due to a more intelligent distribution of data as compared to round-robin. In Fig. 17, we observe that accesses to the memory module from different processors occur within the same interval of time causing bulky arrivals with the round-robin memory management scheme. This problem is alleviated by the improved memory management policies which essentially skew the accesses such that requests arrive in different

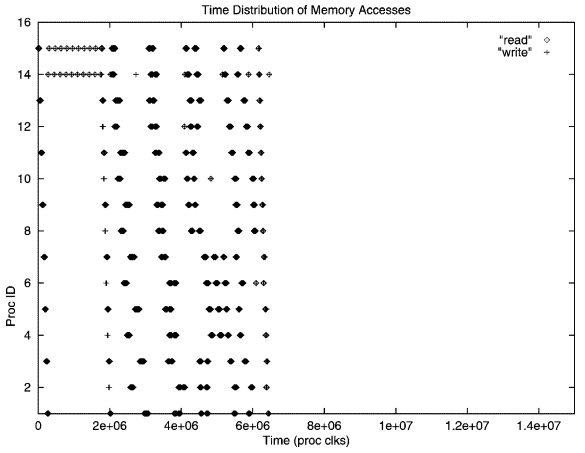

Fig. 19. FFT memory access patterns to memory module 15 using the skew-mapping policy.

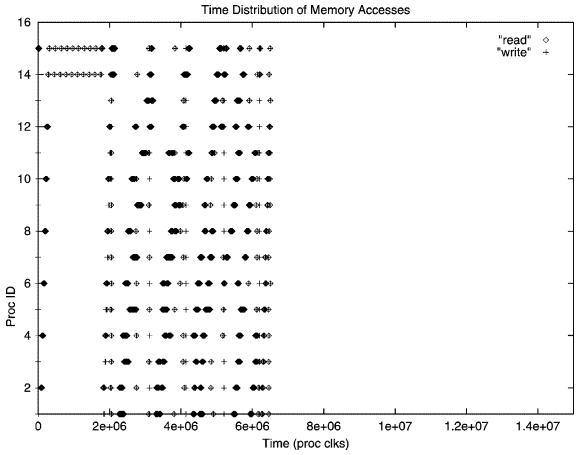

Fig. 20. FFT memory access patterns to memory module 15 using the prime-mapping policy.

time intervals (as seen in Figs. 19 and 20). The per-stage delay for RAW stages is reduced to 1 Mcycles (when compared to 3 Mcycles for buddy), an improvement of over 67%. The overall parallel execution time is reduced to 6.4 Mcycles (when compared to 14 Mcycles for the buddy policy): an improvement of approximately 54%.

For the FFT application, the parallel execution time of the first-touch policy is smaller than that of other memory management policies as shown from Figs. 16–20. Similar observations are made in Fig. 10, where the average latency for memory accesses is provided by first-touch with an interarrival time lower than other memory management policies. The improvement for the first-touch policy is mainly due to: 30% (or 96%) of pages, whose degree of read-sharing (or write-sharing) at page level for the application, is equal to 1. For applications with higher read sharing degrees, we find that skew-mapping and prime-mapping policies perform much better than first-touch.

5. System-centric results

When a processor generates a read or write request, different actions are taken depending on the status of the block in the cache. For example, if it is a read miss and the local memory is the home node, the request is satisfied locally with only

necessary invalidations sent over the network. If the home is a remote node, the request is sent over the network to that node. At the remote node (a) the directory is checked, (b) any necessary invalidations are completed by sending messages over the network, and (c) the data is supplied to the requesting processor over the network again. The paths taken for the forward request and the reply message are different, as shown by bold lines in Fig. 4. These messages encounter delays at various stages of the network. In addition, there is a delay at the interface to pump the message into the first stage of the network, called stage 0. The purpose of this section is twofold: (a) To present and analyze the impact of memory management policies on the stall time of all applications; (b) To measure the effect of improved switch designs and memory management on the waiting delays at different network stages.

5.1. Performance metrics

In this study, we used the application stall time and the average message latency as our primary performance metrics. Stall time is the total amount of time a processor waits for the completion of read/write operations during the parallel execution of the application. We present read, write, memory queue, and invalidation stall times for different applications with varied design parameters for a better understanding of the system overhead on application performance. Stall time is divided into the following important components:

- *Read network latency* (*RNL*): The read network latency includes the waiting time for the network resource and the total time taken in the network to transfer read requests from processor to memory and read replies from memory to processor.
- *Write network latency* (*WNL*): The write network latency includes the waiting time for the network resource and the total time taken in the network to transfer write requests from processor to memory and write replies from memory to processor.
- *Memory queue time* (*MQ*): This is time spent in the memory queue waiting for access to the memory controller that processes the request further.
- *Memory Service Time* (*MS*): This 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. We divided the requests that reach memory into those that require invalidations and those that do not. Thus this time is further divided into:

  - *Memory Service Time w/ Invalidations* (INVMS)
  - *Memory Service Time w/o Invalidations* (NOINVMS)

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 average message latencies for read, write and invalidation are plotted separately, as measured from the simulation. We have further broken down the network latency of a single transaction type into the following to analyze the network thoroughly. Consider the results for a read transaction as an example in Fig. 21(b). The average message latencies for read are denoted by a suffix of _r. 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 stages 0 through 4, as denoted by fw_st0 through fw_st4 for a network with 2×2 switches. With 4×4 switches, the number of stages in the network is reduced by two. Similar to the request message, the time taken for the reply (backward) message over the network is plotted. Since a data block consists of 20 flits, the response transfer time is much bigger, as denoted 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. We finally show the waiting delays at each stage for the backward message (bk_st0 through bk_st4) as explained for the forward message.

5.2. Performance evaluation

We begin our performance evaluation by varying the memory management policies and switch design for all applications. We consider two switch sizes (SW2 for 2×2 and SW4 for 4×4), two switch designs (SWH and BVC32) and the five memory management policies and their effect on stall time and network latency performance. Figs. 21–25 present the stall-time performance in graph: (a) and the average network latency for read transactions in graph, (b) for each of the applications.

In the stall time figures, we observe the great impact of these five memory management policies on the application stall time. For the FFT and FWA applications, the performance using first-touch is better than other memory management policies. The improvements of stall time for the first-touch policy are mainly due to: (1) 30% of pages, whose degree of read-sharing at page level for the FFT application, is equal to 1; (2) about 96% and 94% of pages, whose degree of write-sharing at page level for the FFT and FWA applications, is equal to 1, respectively. This gains great performance for the first-touch policy. For the LU application, there is an enormous reduction (as high as 35%) in stall time using skew-mapping and prime-mapping policies. For the buddy, round-robin, and first-touch policies, the

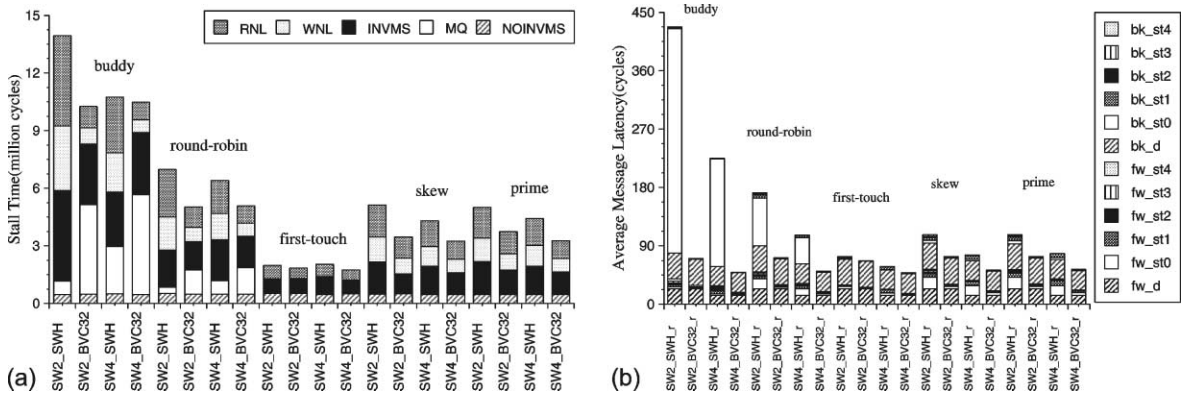

Fig. 21. Effect of memory management policies for the FFT application: (a) Application stall time. (b) Read network latency.

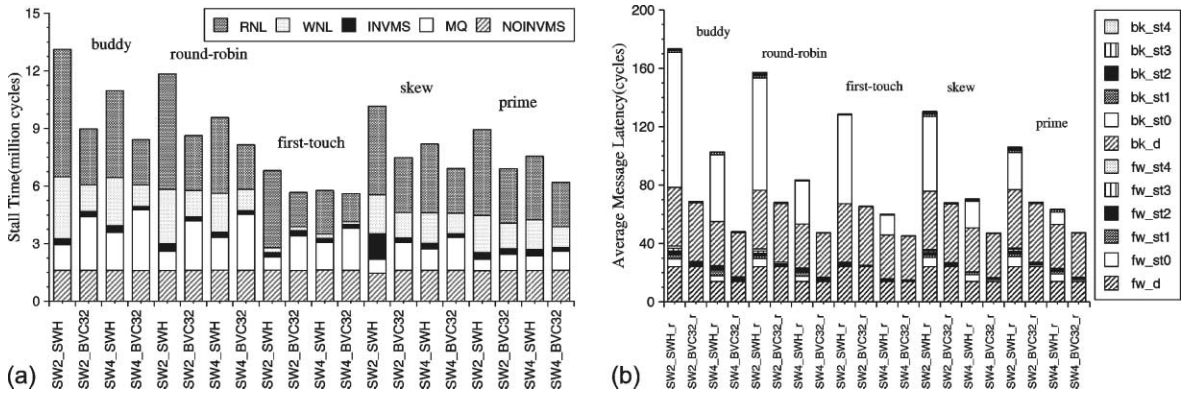

Fig. 22. Effect of memory management policies for the FWA application: (a) Application stall time. (b) Read network latency.

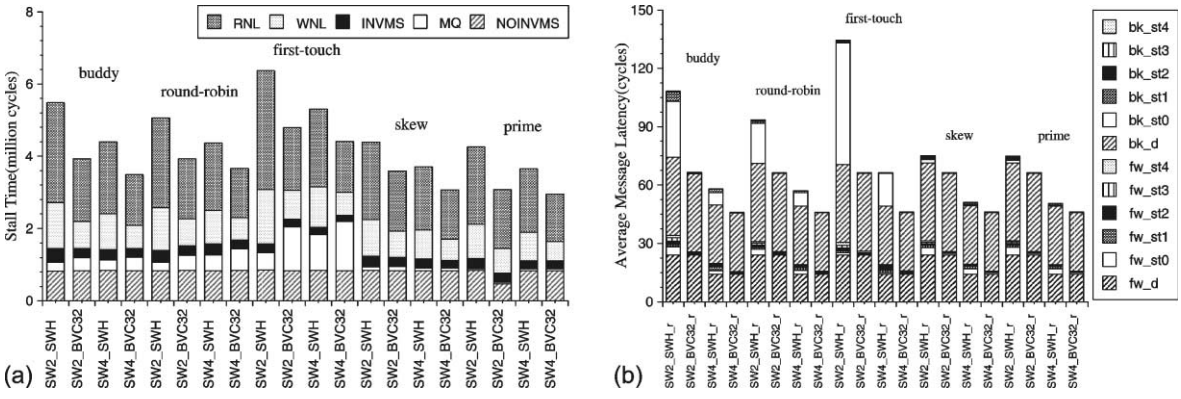

Fig. 23. Effect of memory management policies for the LU application: (a) Application stall time. (b) Read network latency.

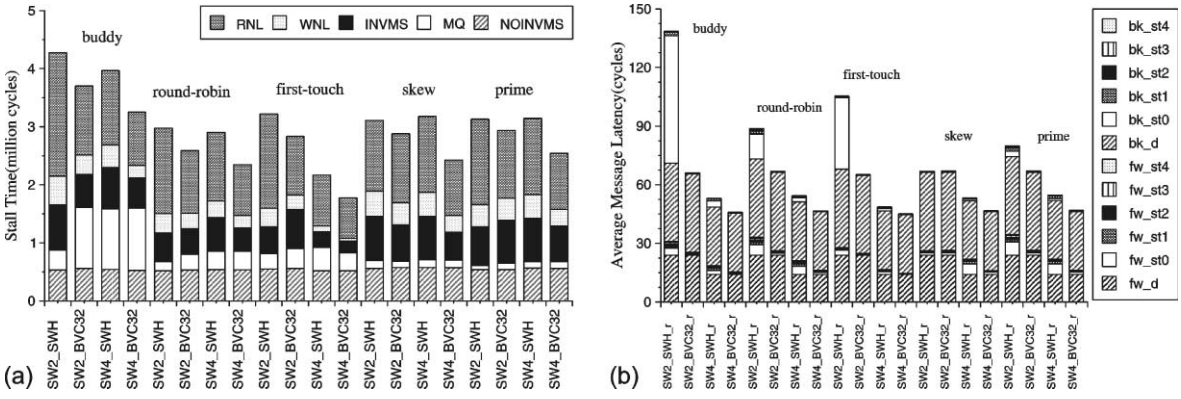

Fig. 24. Effect of memory management policies for the MATMUL application: (a) Application stall time. (b) Read network latency.

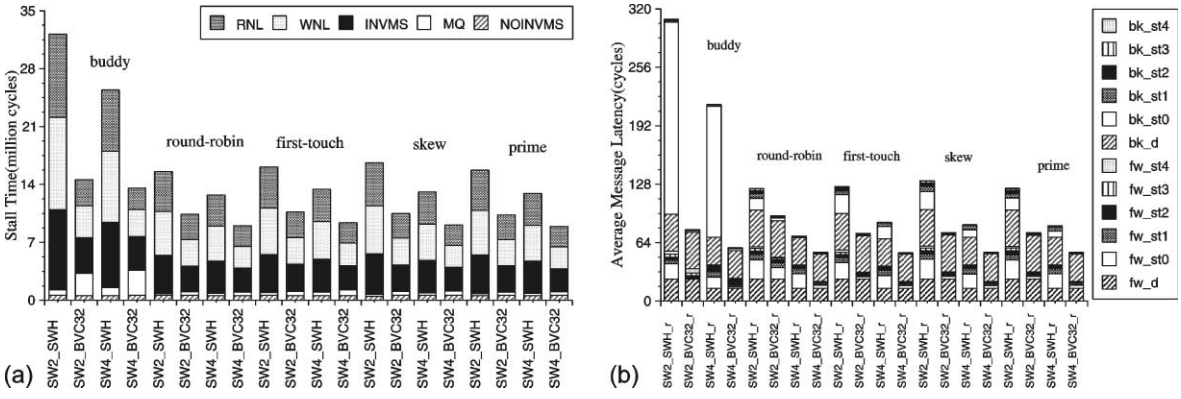

Fig. 25. Effect of memory management policies for the MP3D application: (a) Application stall time. (b) Read network latency.

requests from different processors go to the same memory at the same time giving rise to a bulk of replies (in the form of data blocks) from the same memory. The cause of this bulky response lies in

the distribution of scientific data of size of a power of 2 distributed over a number of memory modules, also a power of 2. The improvements of stall time for the skew-mapping and prime-mapping policies are mainly due to better distribution of the data. This improvement is seen in three out of the five applications. The first-touch policy cannot provide performance benefits because only 1% (8%) of the data pages have a degree of read (write) sharing at page level equal to 1. Of the components of stall time, memory service time without invalidations (NOINVMS) is the least affected by changes in memory management since it comprises of a minor portion of the stall time. On the other hand, read network latency, write network latency, memory queue, and memory service time with invalidations depend highly on the placement of data and improvements in stall-time range from 10% to 90% with an interconnection network employing SWH switches.

Having observed the effect of memory management on the stall time, we now concentrate on the impact of these techniques on the average message latency. The immediate result that can be observed in the average message latency figures is the large amount of delay in the backward stage 0 for buddy and round-robin policies in five applications. For first-touch, the similar phenomenon is seen in three out of five applications. The same situation was observed in the Cedar network [22] 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. With the buddy policy, the data is contiguously stored at a few memory modules causing several processors to send requests to the same memory module at any given time. In round-robin, although the overall requests to the memories are equally distributed over the entire execution period, it is observed that memory requests from many processors are directed to a particular page or multiple pages in the same memory at a certain time. Hence, at a time, the memory or interface has to supply the data blocks to various processors one after another in the form of bulks. For first-touch, each page is located at the processor that first accessed the page. So, the higher the percentage of pages whose degree of data sharing is equal to 1 the more benefit the first-touch policy can gain. The backward stage waiting delay is not observed in skew-mapping and prime-mapping policies since they reduce the amount of simultaneous remote traffic to a single memory module by allocating data intelligently. For these two memory management policies, we see that the average message latency constitutes a smaller fraction of the waiting delay. Much of the delay is due to the transmission delay in the network. Apart from improvements in memory management policy, we also observe that the improvement in switch design (SWH to BVC) reduces the waiting delay significantly at the different stages of the network. The BVC switch architecture along with skew-mapping and prime-mapping memory management policies gives the best performance in most cases.

Improvements in switch design with VC's and buffer sizes of 32 flits increase the complexity of the switch and the cost. However the improvements in performance are also quite impressive. Stall-time improvements between the two networks range from 15% to 65% for the buddy policy, round-robin, first-touch, skew-mapping, and prime-mapping. The exception to this improvement is the performance of the FFT application with the first-touch memory management policy. Improvements due to switch design in this case are minimal since the first-touch policy reduces the number of requests to the network by almost 90% as can be seen in Table 2. With only 10% of the requests traversing the network, switch utilization is low and the SWH switch design is sufficient to provide good performance for FFT with first-touch. Improvements due to increase in switch size are not as impressive as improvements due to the switch design because doubling the switch size effectively reduces the transmission latency by half which does not play an important role in wormhole routing. For the average message latency, another exception to such an improvement is the performance of the MATMUL application with the skew-mapping policy due to the better memory management policy.

6. Conclusion

In this paper, we explored a spectrum of memory management policies for CC-NUMA multiprocessors. We found that existing memory management policies, namely buddy, round-robin and first-touch, each have their own limiting factors for providing good application performance. The problems mainly were due to bulky arrivals of requests to the same memory module. To alleviate these problems, we introduced two improved memory management policies, called skew-mapping and prime-mapping.

The memory access characteristics of five memory management policies, namely buddy, round-robin, first-touch, skew-mapping, and prime-mapping, were analyzed in detail. Our performance metrics covered interarrival time distributions, application access patterns and application stall time. A detailed analysis of the effect of each memory management scheme on application access patterns showed that considerable improvement in performance can be attained by employing intelligent static memory management policies. The impact of memory management on the various components of application stall time was analyzed and shown to be significant. The improvements in stall time for skew-mapping and prime-mapping policies is mainly due to a better distribution of the data among the various memories. The data distribution induces a skewing effect on the temporal access pattern of multiple requests from different processors to the same memory. Improvements were found to be as high as 35% in stall time. For the FFT and FWA applications, the performance using first-touch is better than with the skew-mapping and prime-mapping memory management policies due to low sharing degrees (read or write). However, first-touch is not effective for applications with moderate to high sharing degrees.

The impact of the memory management on network performance was also analyzed and shown to be significant. We used two different switch architectures (SWH and BVC) to represent the advancements in the current interconnect technology. Incorporating buffers and virtual channels in the switch reduces the average message latency tremendously, but we found that performance improvements are very much dependent on the memory management policy.

Acknowledgements

We would like to thank Dr. Hermann Hellwagner and the anonymous referees for their helpful comments.

References

[1] R.P. Larowe, C.S. Ellis, Experimental comparisons of memory management policies for NUMA multiprocessors, ACM Transactions on Computer Systems 9 (4) (1991) 319–363.

[2] C. Scheurich, M. Dubois, Dynamic page migration in multiprocessors with distributed global memory, IEEE Transactions on Computers c-38 (8) (1989) 1154–1163.

[3] J. Ramanathan, L.M. Ni, Critical factors in NUMA memory management, in: Proceedings of the 11th International Conference on Distributed Computing Systems, Arlington, TX, May 1991, pp. 500–507.

[4] W. Bolosky, M. Scott, R. Fitzgerald, R. Fowler, A. Cox, NUMA policies and their relationship to memory architecture, in: Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, CA, April 1991, pp. 212–221.

[5] B. Verghese, S. Devine, A. Gupta, M. Rosenblum, Operating system support for improving data locality on CC-NUMA computer servers, in: Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, October 1996, pp. 279–289.

[6] L.N. Bhuyan, R. Iyer, H. Wang, A. Kumar, Impact of CC-NUMA memory management policies on the application performance of multistage switching network, IEEE Transactions on Parallel and Distributed Systems 11 (3) (2000) 230–246.

[7] D.T. Shapiro, Theoretical limitations on the efficient use of parallel memories, IEEE Transactions on Computers c-27 (5) (1978) 421–428.

[8] D.T. Harper III, Block, multistride vector, and FFT accesses in parallel memory systems, IEEE Transactions on Parallel and Distributed Systems 2 (1) (1991).

[9] A. Deb, Multiskewing—a noval technique for optimal parallel memory access, IEEE Transactions on Parallel and Distributed Systems 7 (6) (1996) 595–604.

[10] M.A. Trenas, J. Lopez, F. Arguello, E.L. Zapata, A memory system supporting the efficient SIMD computa-

tion of the two dimensional DWT, in: Proceedings of International Conference on Acoustics, Speech, and Signal Processing, Seattle, WA, May 1998.
[11] D.H. Lawrie, C.R. Vora, The prime memory system for array access, IEEE Transactions on Computers c-31 (5) (1982) 435–442.
[12] Q. Yang, Introducing a new cache design into vector computers, IEEE Transactions on Computers c-42 (12) (1993) 1411–1424.
[13] T. Sun, Q. Yang, A comparative analysis of cache designs for vector processing, IEEE Transactions on Computers c-40 (3) (1999) 331–343.
[14] L.N. Bhuyan, Q. Yang, D.P. Agrawal, Performance of multiprocessor interconnection networks, IEEE Computer Magazine 22 (2) (1989) 25–37.
[15] W.J. Dally, Virtual channel flow control, IEEE Transactions on Parallel and Distributed Systems 3 (2) (1992) 194–205.
[16] L.N. Bhuyan, R. Iyer, T. Askar, A.K. Nanda, M. Kumar, Performance of multistage bus networks for a distributed shared memory multiprocessor, IEEE Transactions on Parallel and Distributed Systems 8 (1) (1997) 82–95.
[17] M. Ould-Khaoua, H. Sarbazi-Azad, An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic, IEEE Transactions on Parallel and Distributed Systems 12 (3) (2001) 283–292.
[18] H. Sarbazi-Azad, M. Ould-Khaoua, L.M. Mackenzie, Analytical modelling of wormhole-routed *k*-ary *n*-cubes in the presence of hot-spot traffic, IEEE Transactions on Computers c-50 (7) (2001) 623–634.
[19] A. Kumar, L.N. Bhuyan, Evaluating virtual channels for cache coherent shared memory multiprocessors, in: Proceedings of the ACM International Conference on Supercomuting, Philadelphia, PA, May 1996, pp. 253–260.
[20] L.N. Bhuyan, H. Wang, R. Iyer, A. Kumar, Impact of switch design on the application performance of cache-coherent multiprocessors, in: Proceedings of IPPS/SPDP'98, Orlando, FL, April 1998, pp. 466–475.
[21] J.F. Martinez, J. Torrellas, J. Duato, Improving the performance of bristled CC-NUMA systems using virtual channels and adaptivity, in: Proceedings of ACM International Conference on Supercomuting, Rhodes, Greece, June 1999, pp. 202–209.
[22] J. Torrellas, Z. Zheng, The performance of the cedar multistage switching network, IEEE Transactions on Parallel and Distributed Systems 8 (4) (1994) 321–336.
[23] E.A. Brewer, C.N. Dellarocas, A. Colbrook, W.E. Weihl, PROTEUS: A High-Performance Parallel-Architecture Simulator, Technical Report MIT/LCS/TR-516, Massachusetts Institute of Technology, Cambridge, MA, September 1991.
[24] L.M. Censier, P. Feautrier, A new solution to coherence problems in multicache systems, IEEE Transactions on Computers c-27 (12) (1978) 1112–1118.
[25] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, J. Hennessy, The DASH prototype: logic overhead and performance, IEEE Transactions on Parallel and Distributed Systems 4 (1) (1993) 41–61.
[26] G.Astfalk and T.Brewer, An Overview of the HP/Convex Exemplar Hardware, Available from <http://www.convex.com/tech_cache/ps/hw_ov.ps>.
[27] J. Laudon, D. Lenoski, The SGI origin: A ccNUMA highly scalable server, in: Proceedings of the 24th Annual International Symposium on Computer Architecture, Denver,CO, June 1997, pp. 241–251.
[28] M. Galles, Scalable pipelined interconnect for distributed endpoint routing: the SGI SPIDER chip, in: Proceedings of Symposium on High Performance Interconnects (Hot Interconnects 4), Palo Alto, CA, August 1996, pp. 141–146.
[29] J. Carbonaro, F. Verhoorn, Cavallino: The teraflops router and NIC, in: Proceedings of Symposium on High Performance Interconnects (Hot Interconnects 4), Palo Alto, CA, August 1996, pp. 157–160.
[30] J.P. Singh, W.-D. Weber, A. Gupta, SPLASH: Stanford parallel applications for shared-memory, ACM SIGARCH Computer Architecture News 20 (1) (1992) 5–44.
[31] S. Chodnekar, V. Srinivasan, A. Vaidya, A. Siva subramaniam, C. Das, Toward a communication characterization methodology for parallel applications, in: Proceedings of the Third International Symposium on High-Performance Computer Architecture, San Antonio, TX, February 1997, p. 310.

Ravishankar Iyer received his Ph.D. in Computer Science, M.S. in Computer Science and B.S. in Electrical Engineering in August 1999, August 1996 and December 1994 respectively from Texas A&M University, College Station, TX. He is currently with Intel Corporation. His research interests include computer architecture, parallel computing and performance evaluation.

Hujun Wang received his M.S. degree in computer science from NanJing University of Science and Technology, China, in 1993, and B.S. 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.

Laxmi Narayan Bhuyan 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 University of California, Riverside, CA. 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.