Appeared in the IEEE Transactions on Computers, July 1995

Performance Evaluation of the Slotted Ring Multiprocessor

Luiz André Barroso and Michel Dubois

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-2562
(213) 740-9130

barroso@paris.usc.edu
dubois@paris.usc.edu

Abstract: *As microprocessor speeds continue to improve at a very fast rate the bandwidth requirements for system level interconnections in multiprocessors may eventually rule out the use of shared buses even for small scale multiprocessors. On the other hand high-speed unidirectional links are an emerging technology that has the potential to scale with microprocessor technology and could replace buses as the interconnection fabric for future multiprocessors. In this paper we evaluate the performance of the unidirectional slotted ring interconnection for small to medium scale shared memory systems, using a hybrid methodology of analytical models and trace-driven simulations. We use memory traces from actual execution of parallel programs to drive detailed event-driven simulations of a variety of ring and bus multiprocessors. Snooping and directory coherence protocols for the slotted ring are evaluated in the context of multitasking. Snooping is shown to outperform full-map and linked list directory schemes in the unidirectional slotted ring, and it also compares favorably to high-performance split-transaction bus systems.*

Index terms: cache coherence protocols, computer architecture, shared memory multiprocessors, slotted ring, trace-driven simulation, analytical models.

1.0 Introduction and Motivations

Even though much of the research in interconnection networks today aims at connecting thousands of processing elements – the massively parallel processing (MPP) trend – the problem of building interconnections for smaller scale shared memory multiprocessors is still not solved. As new and faster processors are made available each year, it becomes clear that shared buses, the most popular technology for current commercial systems, cannot cope with state-of-the-art RISC microprocessors, such as Digital's 21064 Alpha [8] with a peak advertised performance in excess of 400 MIPS.

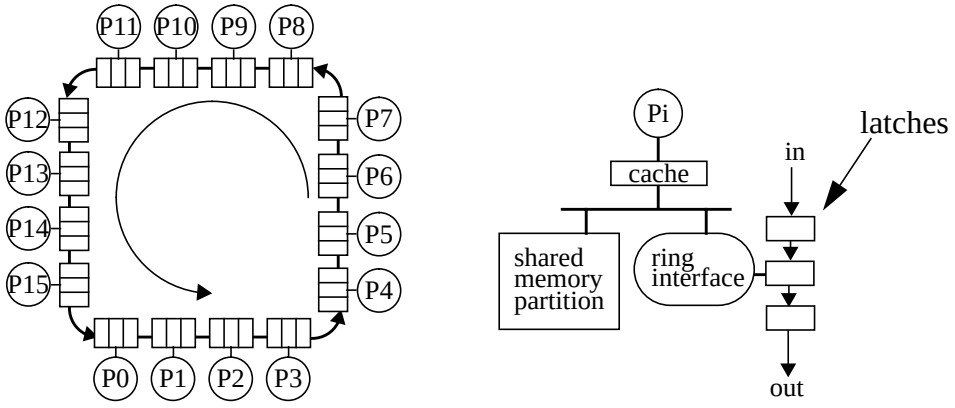
Bus bandwidth is not likely to increase at the same pace as processor and circuit technology improves. The major reasons for the poor technological scalability of bus interconnections are severe problems related to the bus topology itself. First of all, the bus is a mutually exclusive resource which means that only one processor can transmit at any given time. All processors must participate in an arbitration phase before accessing the bus; pipelining is usually limited to the overlap of arbitration and transmission. Another limiting factor is the bus clock cycle in which signals must propagate throughout the entire extension of the bus, and which depends on the length of the bus and its characteristic impedance. Each transmitter on the bus drives several receivers, which results in a higher characteristic impedance and reduces the signal propagation speed. Lastly, since a bus is bidirectional there is an extra delay associated with switching drivers and receivers at the start of each new transfer. Increasing the bus width to transfer more data per bus cycle is an attractive option. However, bus width is constrained by limited pin count and crosstalk interference, and little performance is gained from bus widths surpassing the size of the average transfer.

In the past few years, point-to-point unidirectional connections have emerged as a very promising interconnection technology because they lack most of the problems associated with buses. Point-to-point connection links have only one transmitter and one receiver (one at each end) and can be very short (less than two inches) provided they connect adjacent boards on the backplane. Their characteristic impedance is very small and, if they are terminated properly, the transmission speed is much higher than for buses. An additional factor favoring point-to-point transmission is signal pipelining. A new transmission can be started before the previous one has reached the receiver. Therefore clocking speed and throughput are not limited by wire length. Overall, point-to-point connections are more technologically scalable than bus connections, and we can expect their delivered bandwidth to benefit continuously from improvements in circuit technology. The potential of point-to-point communications in shared memory multiprocessors is

demonstrated by the IEEE Scalable Coherent Interface (SCI) [12] set of standards, based on 500 MHz 16-bit wide point-to-point links in its first generation of circuits.

In this paper we evaluate the unidirectional slotted ring as an alternative to buses for cache-based multiprocessor systems with up to 64 processors and in the context of multitasking. Snooping and directory protocols are first described for the slotted ring, and then their performance is compared using a hybrid methodology including trace-driven simulations and analytical models. An earlier version of this work appeared in [2]. This paper extends [2] in several ways, which includes: addition of PTHOR to the list of benchmarks; use of compiled optimized code (option -O2 of gcc) in the trace generation and subsequent re-run of all evaluation experiments; inclusion of a subsection to describe the analytical models in detail; addition of a subsection that shows the performance evaluation of linked list protocols with respect to full-map directory protocols; inclusion of a subsection that shows the effects of block size in the performance of the slotted ring multiprocessor; expansion of the subsection that compares slotted rings and buses through the addition of two more benchmarks, which doubles the amount of results presented in [2]; overall detailing and expansion of the discussions of performance results.

FIGURE 1. The Unidirectional Ring Interconnect



(a)
A 16-Node Slotted Ring

(b)
Node Structure

The slotted ring architecture is described in the next Section. Section 3 briefly explains the snooping cache coherence protocol as proposed in [1] as well as a directory-based protocol, both for the slotted ring architecture. Our evaluation methodology is explained in Section 4. Quantitative evaluations of the two cache coherence strategies for the slotted ring are shown in Section 5, which also includes a comparative performance analysis of the slotted ring with a high-end split-transaction bus. Related work is discussed in Section 6. Final remarks and conclusions are drawn in Section 7.

2.0 The Slotted Ring

The unidirectional ring is the simplest form of point-to-point interconnection, which means minimum number of links per node and simpler interface hardware. In particular, the unidirectional ring requires the simplest routing mechanism possible: the only routing decision is whether to remove a message from the ring or to forward it to the next node. Consequently store-and-forward is avoided; communication delays are shorter and the raw bandwidth provided by point-to-point links is better utilized. Point-to-point connections are so fast that the board logic can eventually become the performance bottleneck, and therefore simple and fast routing mechanisms will be critical.

The general architecture of the unidirectional ring is shown in Figure 1, and consists of a set of processing elements containing a CPU, local cache memory, a fraction of the shared memory space, and a ring interface. The data path on the ring interface consists of one input link, a set of latches, and one output link. At each ring clock cycle the contents of a latch are copied to the following latch, within a ring interface and across the links, so that the interconnection behaves as a circular pipeline. The main function of the latches is to hold an incoming message for a few clock cycles in order to determine whether to forward it or not. The number of latches in each interface should be kept as small as possible so to reduce the latency of messages.

The ring access control mechanism which dictates when a node can send a message is complicated by the fact that messages can be larger than the width of the data path, and may span multiple pipeline stages. Furthermore, messages can have different sizes. An interconnection for a cache-coherent system has to deal with at least two types of messages, which we call *probe messages* (or probes) and *block messages* (or blocks). Probes are short messages carrying miss and invalidation requests, consisting typically of a block address field and other control/routing information. Block messages are made up of a header, which is similar to a probe, plus a cache block, and carry cache blocks for misses and write-backs.

To avoid the complexities associated with breaking messages into packets, the protocol must transmit messages in consecutive pipeline stages. Therefore, consecutive empty stages to fit a particular message must be found and arbitrated for in a distributed fashion. There are three main solutions for the ring access control problem: token passing rings, register insertion rings and slotted rings. In token passing rings a special bit pattern, called token, is passed from node to node allowing the node with possession of the token to transmit. The main disadvantage of token passing is that only one message may travel on the ring at a time, which is a waste of bandwidth if the ring can accommodate more than one message. In the register insertion approach, chosen for

the SCI standard, a bypass FIFO between the input and output stages of the ring interface buffers incoming messages while the local processor is transmitting. When the transmission is completed, the contents of the FIFO are forwarded to the output link and the local processor is not allowed to transmit until the FIFO is emptied.

In the slotted ring approach the ring is divided into marked message slots with different sizes circulating continuously through the system. A processor ready to transmit a message waits until an empty slot with the same size as the message passes through. A single bit in the header of the slot identifies an empty slot. The slotted ring restricts the utilization of the ring bandwidth by different message types because the mix of message slots is pre-determined. This restriction has no impact on performance provided the mix of slots matches the expected mix of messages. The particular mix depends on the cache coherence protocol, and therefore we postpone this discussion until the next Section. A slotted ring can be seen as a generalization of a token ring in which slots represent the tokens.

Bhuyan et al [4] use an analytical framework to compare the performance of slotted and register insertion rings. Their overall conclusion is that under light loads the register insertion ring may exhibit faster access times than a slotted ring since a message does not wait for a proper slot to pass by. On the other hand, under medium to heavy loads, the simplicity of enforcing fairness on the slotted ring may yield a better performance than in the register insertion approach. Bhuyan also notes that the differences in performance are relatively small therefore very sensitive to the parameters of a particular design. We also believe that it is simpler to implement a slotted ring interface because it does not require a bypass buffer. We chose the slotted ring in our evaluations mainly for its simplicity, but also because it can support both snooping and directory protocols. The register insertion approach is not suited to a snooping protocol. An implementation of snooping on a token ring is presented by Delp and others in [7].

3.0 Coherence Protocols for the Slotted Ring

3.1 A Snooping Protocol

It is generally believed that snooping protocols are only suitable for bus-based systems, and therefore protocols based on directories are favored [5,6] for point-to-point connected systems, such as the slotted ring. This intuition is based on the fact that snooping relies heavily on the broadcast of coherence requests, which comes for free in bus systems but can be very expensive in general point-to-point interconnects. We contend however that snooping is an attractive strategy for the unidirectional ring due to its low cost of broadcast with respect to unicast. The

bandwidth used by a broadcast in an unidirectional ring is roughly twice the bandwidth used for a unicast. Moreover, only probes have to be broadcast; block messages, which are generally longer, do not require broadcasting. Snooping protocols require less state information at the memory modules and are less complex than most protocols.

The snooping cache coherence protocol for the slotted ring, introduced in [1], is a write-invalidate write-back protocol, logically similar to an ownership-based snooping protocol for a split-transaction bus. Three cache states, *Invalid* (INV), *Read-Shared* (RS), and *Write-Exclusive* (WE), indicate whether the block is not present in the cache, present in read-only mode, or present in read-write mode respectively. The node to which the address of a cache block maps is called the *home* node for that block. The node that has a WE copy of a block is called the *dirty* node. A *dirty bit* is kept per block frame in the home node to indicate when a block is cached in WE state. The dirty bit is very important to the snooping implementation on the ring since it indicates to the home node whether it owns a block, i.e., whether it should respond to miss requests for that block. When the dirty bit is set the dirty node is the owner and therefore responsible for responding to coherence requests. When there is no dirty node the dirty bit is reset and the home node responds.

TABLE 1. Snooping Protocol

Transaction	Protocol Behavior

Read Miss	if <block is dirty>
	  dirty node sends copy to requester & changes to RS
	  requester receives block & forwards copy to home node
	  home node resets dirty bit
	else
	  home node sends copy to requester
	requester final state is RS

Write Miss	if <block is dirty>
	  dirty node sends copy to requester & changes to INV
	else
	  all caches with RS copy change to INV
	  home node sends copy to requester & sets dirty bit
	requester final state is WE

Invalidation	/* block is already cached in RS state at requester */ all nodes with RS copies change to INV home node sets dirty bit

Read miss, write miss, and invalidation[1] requests are broadcast through the ring in probe slots. Probes are inserted and removed by the requester and are “snooped” as they pass through each node in the system. The node with the valid block copy (the *valid* node) acknowledges a

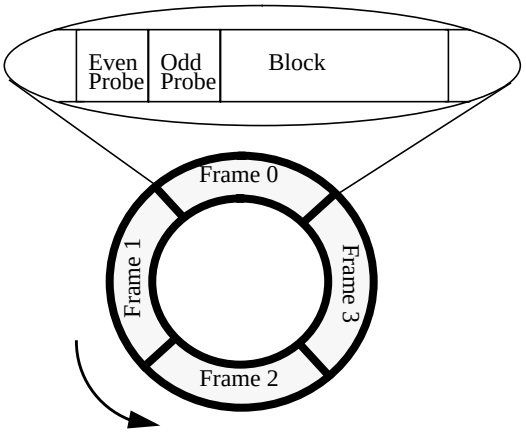
1. The difference between a write miss and an invalidation is that an invalidation is issued when the node already has a RS copy of the block, and it is only requesting permission to write.

probe message. Coherence actions at the snooper are consistent with typical write-invalidate protocols.

The most important feature of this snooping protocol is that no coherence transaction traverses the ring more than once. In this sense, the latency of coherence transactions is minimum. This is possible because all probes are snooped on as they pass through each ring interface, and are only removed from the ring by the requester. The valid node does not send the acknowledgment in the same probe slot, but rather in an acknowledgment field in the following probe slot of the same type. Furthermore, the latency of misses is independent of the relative positions of the requesting node and of the valid node. Therefore, the slotted ring with a snooping protocol behaves as a uniform memory access (UMA) interconnect, just like a shared bus.

Snooping implementations have harder real-time constraints than non-snooping implementations, since the snooper must react to all remote memory operations issued in the system. With today's point-to-point connection speeds of up to 500 MHz, the snooper hardware could not respond at the maximum rate of incoming probes. That prevents snooping from being implemented on a register insertion ring. The slotted ring access control mechanism proposed in [1] is one way to overcome this problem. In this scheme probe slots on the ring are separated by a minimum number of clock cycles by interleaving them with other types of slots, forming *frames*. A frame is composed of one probe slot for even-address blocks, one probe slot for odd-address blocks, and one block slot (Figure 2).

FIGURE 2. Dividing the slotted ring into frames



Therefore, if the dual-directory on the ring interface is 2-way interleaved, two consecutive probes for the same dual-directory bank are always spaced by at least a frame size, which is no less than 20 nsec for a 32-bit wide ring using a 16-byte cache block size and clocked at 500 MHz (2 nsec). Table 2 displays the probe inter-arrival times for various ring widths and block sizes,

considering a 2-way interleaved dual-directory and 500 MHz ring links. The mix of 2 probe slots per block slot is the optimum for the snooping protocol, because the number of probes and block messages generated in actual simulations is approximately the same, and probes traverse the whole ring whereas block messages are removed by the destination and traverse only half the ring on the average.

TABLE 2. Snooping rate (nsec.)

block size	ring data width (bits)		
	16	32	64
16 bytes	40	20	10
32 bytes	56	28	14
64 bytes	88	44	22
128 bytes	152	76	38

3.2 A Full-map Directory Protocol

The full-map directory protocol outlined here has the same cache states as the snooping protocol. The home node knows whether the block is dirty, and which nodes currently have valid copies. A directory entry is kept with each memory block with one presence bit for each node in the system plus one dirty bit. A set presence bit indicates that the node may have a copy of the block. All coherence requests are first sent to the home node. The home node looks up the directory entry for the block and takes the appropriate coherence actions following Censier and Feautrier's scheme [5].

Read misses on clean blocks take only one trip around the ring, since it is a request/ response transaction between the requester and the home node only. The latency in this case is the same as in the snooping protocol. Whenever the home node is not the owner, a request is forwarded to the dirty node. If the dirty node is on the path between the requester and the home, one extra trip around the ring is needed, as seen in Figure 3.b. Moreover, on all write misses and invalidations for blocks that are cached RS elsewhere, the home node must send a multicast invalidation and wait for the reply before responding; this case also requires one extra ring traversal Furthermore, we assume that the directory protocol is optimized in such a way that the dirty node can reply directly to the requester and concurrently send the acknowledgment to the home node. The alternative would be to make the dirty node reply to the home node, which would then reply to the requester. In this case all misses to dirty blocks would require two complete ring traversals. Another optimization included here is to assume that the home node is capable of sending a single multicast message when it has to invalidate multiple copies of a block, instead of

having to send multiple messages. For the remaining of this paper whenever we use ‘directory protocol’ we will be referring to the full-map directory scheme.

TABLE 3. Full-Map Directory Protocol

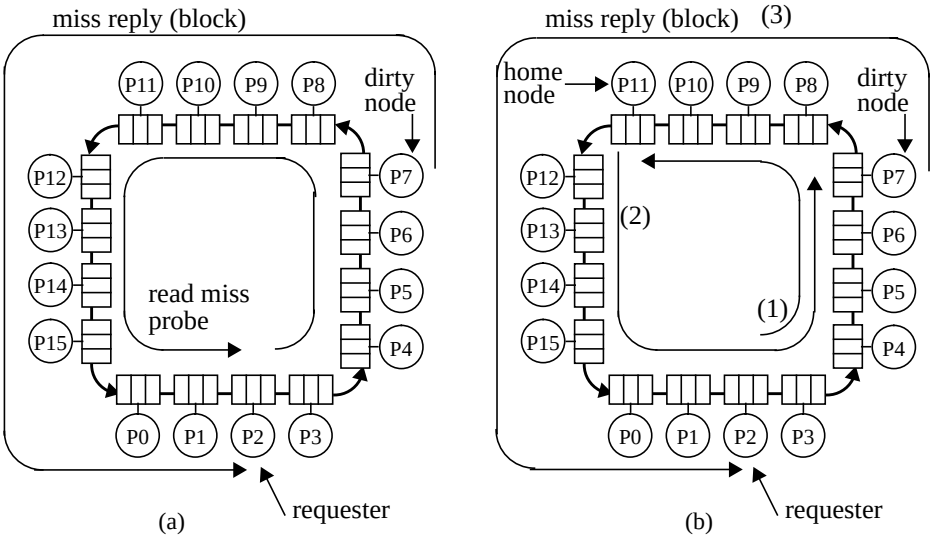
Transaction	Protocol Behavior

Read miss	if <block not dirty>
	  home node records requester as having RS copy
	  home node sends copy to the requester
	else
	  home node forwards request to the dirty node
	  dirty node changes its state to RS & sends copy to requester
	  requester stores block as RS and forwards it to home node
	  home node records the two nodes as having RS copies

Write miss	if <block uncached>
	  home node records requester as having the WE copy
	  home node sends copy to requester
	if <block cached not dirty>
	  home node sends invalidation to nodes with RS copies
	  & sends copy to requester; records it as having the WE copy
	if <block cached dirty>
	  home node forwards request to dirty node
	  dirty node replies to home node and sends copy to requester
	  dirty node changes to INV; requester changes to WE
	  home node records requester as having the WE copy

Invalidation	if <no other cached copies of the block>
	  home node records requester as having the WE copy and acks req.
	else
	  home node sends invalidation to all nodes with RS copies
	  home node records the requesting node as having the WE copy

FIGURE 3. Read miss on a dirty block: (a) Snooping vs. (b) Directory



3.3 Linked List Protocol

Linked list protocols have been adopted by the SCI standard [12] and thus deserve special attention. In a linked list protocol, each block frame in a cache has one or more pointer fields linking all nodes with cached copies of a block in a *sharing list*. The home node keeps a pointer to the node at the head of the sharing list (the *head* node), which is responsible for maintaining the coherence of the block. For some requests, the unidirectional ring under a linked list protocol must be traversed multiple times. Since the head node is responsible for the coherence of the block, the home node must forward the request to the head node on each miss to a block currently cached. As in the full-map protocol, the ring may be traversed once or twice, depending on the relative positions of the requester, the home and the head. Note that unlike the full-map protocol, multiple ring traversals will occur even in read misses for non-dirty (but cached) blocks. Furthermore, invalidating the sharing list takes extra ring traversals when the order of the nodes in the sharing list conflicts with the direction of the ring. In the worst case, it may take *n* traversals to invalidate a block shared by *n* nodes.

Besides linked list protocols, other directory protocols such as limited directory protocols [6] have been proposed. In general, linked list and limited directory protocols save memory and reduce directory contention especially in large scale systems, but they are not likely to outperform full-map directories in the context of smaller scale systems such as the slotted ring. Therefore we mostly compare full-map directories to snooping in this paper. In Section 5.1.2 we present simulation results that compare the performance of linked list and full-map directory protocols in the context of the slotted ring.

3.4 Discussion

The performance differences between the snooping and the full-map directory protocol is very dependent on sharing patterns. If an application has very little read-write sharing of blocks, the vast majority of misses to shared data are misses on clean blocks, and most invalidations find the block uncached elsewhere. In this case the latencies for the full-map and the snooping protocols are similar and the full-map protocol needs less bandwidth than the snooping protocol because the probes are not broadcast in the full-map scheme. Therefore the directory protocol could outperform snooping because of the lower contention experienced by probe messages. On the other hand, in applications with a fair amount of read-write block sharing, the coherence transactions in the directory protocol may experience significantly non-uniform and higher latencies. This non-uniformity of latencies cannot be easily avoided by intelligent placement of data or scheduling of processes. For example, in Figure 3.b, if processors *P2* and *P7* share a block

read-write, the configuration depicted always occurs for either *P2* or *P7*, no matter where the home node of the block is (unless it is *P2* or *P7*).

The optimum mix of probe and block slots is not as predictable for the directory scheme as it is for snooping because transactions do not always commit in a single ring traversal and may require one or two probe messages. For the benchmarks under study the number of probe messages generated during program execution was typically between 1 and 2 times the number of block messages. We then performed simulations using probe to block ratios of 1:1 and 2:1, and it was determined that a 2:1 ratio was also the best choice for the directory scheme.

4.0 Evaluation Methodology

We have quantitatively compared the performance of the slotted ring under different classes of coherence protocols and of a high-performance split-transaction bus under a snooping protocol. The evaluation methodology is a hybrid one, composed of trace-driven simulations and analytical models and similar to prior studies [6,14]. Very detailed trace-driven simulations of a limited number of configurations were first performed to gain better understanding of the interactions between the programs and the architectures. Then a set of simple analytic models were formulated to capture the essential performance characteristics of the slotted ring multiprocessor and a packet switched bus system. We have built models of the snooping and the full-map directory protocol for the slotted ring, and for a snooping protocol for the bus. For the comparison of snooping and full-map directory protocols for the slotted ring, as well as the comparison between a snooping slotted ring and a snooping bus system, we used the following methodology:

(1) A detailed trace-driven simulation of the particular protocol and architecture for one particular choice of values for processor and interconnection speed was performed. This generated a variety of performance parameters including counts for all significant system events.

(2) The event counts generated by the simulation were used to parameterize the analytical models which were then used to extrapolate performance results for other values of processor and interconnection speeds.

(3) We ran the trace-driven simulator for a few other processor and interconnection values and verified the accuracy of the extrapolations from (2).

The advantage of this hybrid approach is that we have the type of confidence in the accuracy of the results that is expected from detailed simulations, but we also have the agility of analytical models to explore the design space more thoroughly and efficiently. Each run of our

trace-driven simulations takes an average of 6-8 CPU hours to complete, on a Sun SPARCStation2, generating only one point in the design space for each benchmark. The analytical models typically take less than a second of CPU to generate complete curves on the same platform. The analytical models are described in Section 4.2.

In Sections 5.1.2 and 5.1.3 we did not use any analytical models, only detailed trace-driven simulations. This was possible since we were only interested in a few performance points.

4.1 Benchmarks

The models and simulations are driven by two sets of benchmarks. The first set is a group of three programs from the Stanford SPLASH benchmark suite [18]: MP3D, WATER, CHOLESKY and PTHOR. These programs were traced using the CacheMire simulator [3] and traces were obtained for systems with 8, 16 and 32 processors. Traces for the second set of benchmarks are 64-processor traces of three parallel FORTRAN programs: FFT, WEATHER and SIMPLE [6].

MP3D is a rarefied fluid flow simulation program used to study the forces applied to objects flying in the upper atmosphere at hypersonic speeds, and it is based on Monte Carlo methods. WATER evaluates the interactions in a system of water molecules in liquid state and consists of solving a set of motion equations for molecules confined in a cubic box for a number of time steps. CHOLESKY performs a parallel Cholesky factorization of a sparse matrix, and it uses supernodal elimination techniques. PTHOR is a digital circuit simulator that uses a variant of Chandy-Misra distributed time algorithm with deadlock resolution. FFT is a radix-2 fast Fourier transform program. SIMPLE solves equations for hydrodynamics behavior using finite difference methods. WEATHER also uses finite difference methods to model the atmosphere around the globe.

The main characteristics of the traces derived from each benchmark are shown in Table 4. For a given program, the number of references tends to increase as it is mapped to larger multiprocessor configurations. That increase is due to instruction blocks associated with thread initialization and management. The increase is most noticeable in CHOLSEKY and PTHOR since they use queue based thread scheduling instead of static scheduling. The miss rate values were derived with 128 KBytes direct-mapped data caches with a block size of 16 bytes. We further assumed that instruction references never miss, in order to reduce the simulation times. This assumption did not impact the results since the actual hit rate in the instruction cache is extremely high. The analysis also assumes that a processor blocks on all misses and invalidations, and the time to access a local memory bank is fixed at 140 nsec. for all systems. The simulations of the

ring and bus systems were developed using the CSIM package [17] which is a library of C functions for process oriented simulation. The above assumptions remain fixed throughout the rest of the paper unless it is explicitly said otherwise.

TABLE 4. Basic Trace Characteristics (miss rates derived for 16B blocks, 128 KB caches)

benchmark	proc.	data references ($x10^6$)	instruction references ($x10^6$)	% private data references	% shared reads	% shared writes	total data miss rate	shared data miss rate
MP3D	8	4.10	10.90	28.3 %	44.6 %	26.9 %	7.31 %	10.01 %
	16	4.25	11.52	27.4 %	46.3 %	26.0 %	7.85 %	10.61 %
	32	4.74	13.60	24.7 %	50.9 %	23.3 %	16.89 %	22.21 %
WATER	8	5.18	9.72	78.6 %	19.1 %	2.15 %	0.42 %	1.84 %
	16	5.31	10.22	76.8 %	20.8 %	2.11 %	0.65 %	2.50 %
	32	5.44	10.76	74.6 %	22.5 %	2.07 %	1.39 %	5.18 %
CHOLESKY	8	2.36	7.02	36.6 %	51.0 %	9.93 %	8.55 %	12.01 %
	16	3.17	9.92	35.6 %	53.8 %	7.54 %	16.38 %	23.29 %
	32	5.19	17.5	32.1 %	59.1 %	4.67 %	35.73 %	50.16 %
PTHOR	8	15.8	51.4	25.9 %	67.6 %	5.18 %	5.17 %	6.84 %
	16	22.6	74.8	21.7 %	73.2 %	3.91 %	4.65 %	5.89 %
	32	39.5	131.3	18.6 %	77.7 %	2.55 %	5.32 %	6.50 %
FFT	64	4.31	3.12	76.0 %	12.0 %	11.9 %	6.85 %	26.12 %
WEATHER	64	15.63	13.64	83.9 %	13.0 %	3.09 %	5.25 %	30.78 %
SIMPLE	64	14.02	11.59	70.9 %	25.9 %	3.17 %	15.97 %	54.16 %

4.2 Analytical Models

We briefly describe our approach to model the performance of applications executing on a slotted ring multiprocessor. Our model is based on the methodology proposed by Menasce and Barroso in [14]. It is an iterative approximate method in which a performance model of the program generates input parameters to solve a performance model of the interconnection, which in turn provides new parameters for the model of the program. This is a hybrid methodology, since the performance model for the program receives as inputs some statistics from trace-driven simulations of the program under study. The parameters and formulas used here consider only the snooping protocol for the slotted ring. The model for the full-map directory protocol is very similar but uses slightly different parameters, since parameters reflecting the intensity of read-write sharing have to be taken into account.

The model for the program derives the execution time based on the statistics collected from trace-driven simulation, from basic timings of the architecture, and from estimates of the network contention. The statistics derived from trace-driven simulations are listed in Table 5.

TABLE 5. Parameters from trace-driven simulations of the program

Parameter	Definition
N_{cyc}	Total number of instructions executed in a given processor
N_{lmiss}	Total number of misses to local memory by a processor
N_{smiss}	Total number of misses to shared memory by a processor
N_{inv}	Total number of invalidations sent by a processor
N_{wback}	Total number of write-backs by a processor

The timing parameters from the architecture are the time to satisfy a hit, the time to satisfy a local miss (L_{lmiss}) and the clock cycle of the interconnection. Here we assume that the private cache can respond for hits in one processor cycle (*Pcyc*), and that a hit in the data cache can be overlapped with an instruction fetch. Therefore, the execution time of the program, considering homogeneous execution in all processor nodes is given by

$$PET = N_{cyc}*Pcyc + N_{lmiss}*L_{lmiss} + N_{shmiss}*L_{smiss} + N_{inv}*L_{inv} \quad \text{(EQ 1)}$$

In Equation 1 above, the only unknowns are the latency of misses to shared memory (L_{smiss}) and the latency of invalidations (L_{inv}). These parameters depend not only on the pure latency of remote accesses but also on the contention level in the interconnect, which have to be determined from a performance model of the interconnection. A performance model of an interconnection generally receives as input the mean access rate by each processor. Since we are considering a shared memory environment in which messages are either probes (miss requests/ invalidation requests) or block messages (messages that carry a cache block), it is important to determine the rates of probes and block messages. These rates can be obtained from Equations 2 and 3 as follows

$$\lambda_{probes} = \frac{N_{smiss} + N_{inv}}{PET} \times N_{proc} \quad \text{(EQ 2)}$$

$$\lambda_{block} = \frac{N_{smiss} + N_{wback}}{PET} \times N_{proc} \quad \text{(EQ 3)}$$

where N_{proc} is the number of processors in the system. We now derive a simple performance model of the slotted ring similar to the one used by Bhuyan and others in [4]. The throughput of the slotted ring for probes and block messages is as follows:

$$\mu_{probes} = \frac{P_{slots}}{pipesize \times R_{clock}} \quad \text{(EQ 4)}$$

$$\mu_{blocks} = \frac{B_{slots} \times 2}{pipesize \times R_{clock}} \qquad \text{(EQ 5)}$$

where P_{slots} and B_{slots} denote the number of probe and block slots respectively, *pipesize* is the number of pipeline cycles in the slotted ring, and R_{clock} is the ring clock cycle time. The factor of two in the numerator of Equation 5 comes from the fact that a block message travels only half the ring, on the average. From Equations 2 to 5 we can calculate the average slot utilizations for probe and block messages. From this point on we proceed only with the derivation for probe messages, since the one for block messages is identical. The utilization of probe slots is given by Equation 6 and it can be seen as the probability that a given slot will be busy, i.e., not available.

$$U_{probes} = \frac{\lambda_{probes}}{\mu_{probes}} = 1 - p_0 \qquad \text{(EQ 6)}$$

When a processor has a message ready to send it waits until the first appropriate slot passes through it. If we assume that the arrival process is Poisson distributed, the residual time to find the beginning of a slot can be considered to be uniformly distributed between *0* and T_{frame}, where T_{frame} is the time interval between two slots of the same type. Therefore we can write the average waiting time to find an empty probe slot as

$$W_{probes} = T_{frame} \times \left(1/2 + \sum_{i=1}^{\infty} i \times U_{probes}{}^{i} \times (1 - U_{probes}) \right) \qquad \text{(EQ 7)}$$

which reduces to

$$W_{probes} = T_{frame} \times \left(1/2 + \frac{U_{probes}}{1 - U_{probes}} \right) \qquad \text{(EQ 8)}$$

Notice that there is no queueing effect in each node since we consider that a processor can have at most one outstanding request pending at any time. Now, the unknown terms in Equation 1 can be calculated as follows

$$L_{smiss} = W_{probes} + R_{clock} \times pipesize + L_{lmiss} + W_{blocks} \qquad \text{(EQ 9)}$$

$$L_{inv} = W_{probes} + R_{clock} \times pipesize \qquad \text{(EQ 10)}$$

We start by setting W_{probes} and W_{blocks} to zero, and then iterate until the relative difference observed between successive iterations is less than a given threshold. Menasce and Barroso [14] proved that as long as the latency of interconnection accesses is a monotonically non-decreasing function of the request rate, this iteration converges. The convergence of our particular implementation of this method is very fast, seldom requiring more than 10 steps. A similar model was derived for the performance of a packed-switched bus. The model predictions for processor and ring slot utilizations show very good accuracy for the 8, 16 and 32 processor configurations. Predictions of message latency are not as accurate; nevertheless they still capture the general trends of the simulated curves. The split-transaction bus model was also very accurate in predicting utilization and latency parameters with respect to the results of the simulations.

5.0 Performance Results

Most of the evaluations discussed in this section are derived using the 3-step methodology described in Section 4.0. All model predictions fell within 15% of the simulated values for latencies, and within 5% for processor and network utilizations. Only Figures 4, 8, 9, 10 and 11 use values obtained directly from the trace-driven simulations. We use 128 KBytes direct-mapped data caches with a block size of 16 bytes in all evaluations with the exception of the ones shown in Figure 11, in which the effect of different block sizes is studied.

5.1 Snooping vs. Directory Protocols for the Slotted Ring

The comparison between the snooping and the directory protocols for the slotted ring is shown in Figures 4 to 8. Figure 4 shows a breakdown of the shared data misses for directory into *local*, *remote clean*, *1-cycle dirty* and *2-cycle* misses. *Local* misses are shared data misses that can be satisfied within the requesting node, so that no messages have to be sent on the ring. *Clean* misses are misses to non-dirty blocks mapping to a remote home, taking only one ring traversal and involving one probe message and one block message; *1-cycle dirty* misses are misses to dirty blocks that also require only one ring traversal because of the fortunate relative position of the dirty node with respect to the requester and the home node, but take longer than clean misses because they require 3 hops instead of 2; *2-cycle* misses are the remaining shared misses taking two ring traversals (as shown in Figure 3.b). Processor utilization[1] and average ring slot utilization are displayed for systems with 8, 16 and 32 processors for the SPLASH benchmarks (Figure 5), and for systems with 64 processors for the remaining benchmarks (Figure 7). The differences in the latency of misses between snooping and directory is shown in Figure 6 using MP3D and CHOLESKY. The clock rate of the ring is fixed at 500 MHz (2 nsec) -- 32-bit wide

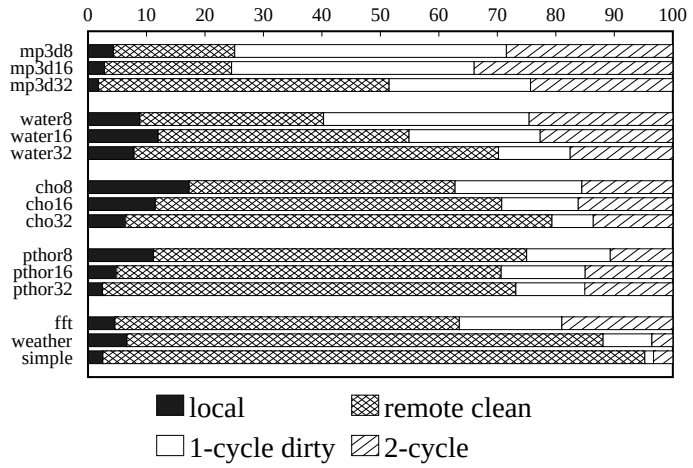
1. Processor utilization is the fraction of time that the processor is busy, instead of waiting for misses or invalidations.

links -- and the processor cycle time varies from 1 to 20 nanoseconds. The simulations assume a scalar RISC-like single cycle instruction execution while accesses hit in the cache. Therefore, a processor cycle of 20 nsec. means 50 MIPS of processing power. Using a 2 nsec. ring clock cycle the pure round-trip latency for an 8 processor ring is 50 nsec (excluding contention) and it increases linearly with the number of nodes.

5.1.1 Discussion

We observe from Figure 4 that the fraction of remote clean misses tends to increase with the system size for each of the SPLASH benchmarks. The increases on the fraction of remote clean misses seem to follow the behavior of the data miss ratio (Table 4). In other words, it appears that whenever the miss ratio increases, most of the added misses are to clean or uncached blocks.

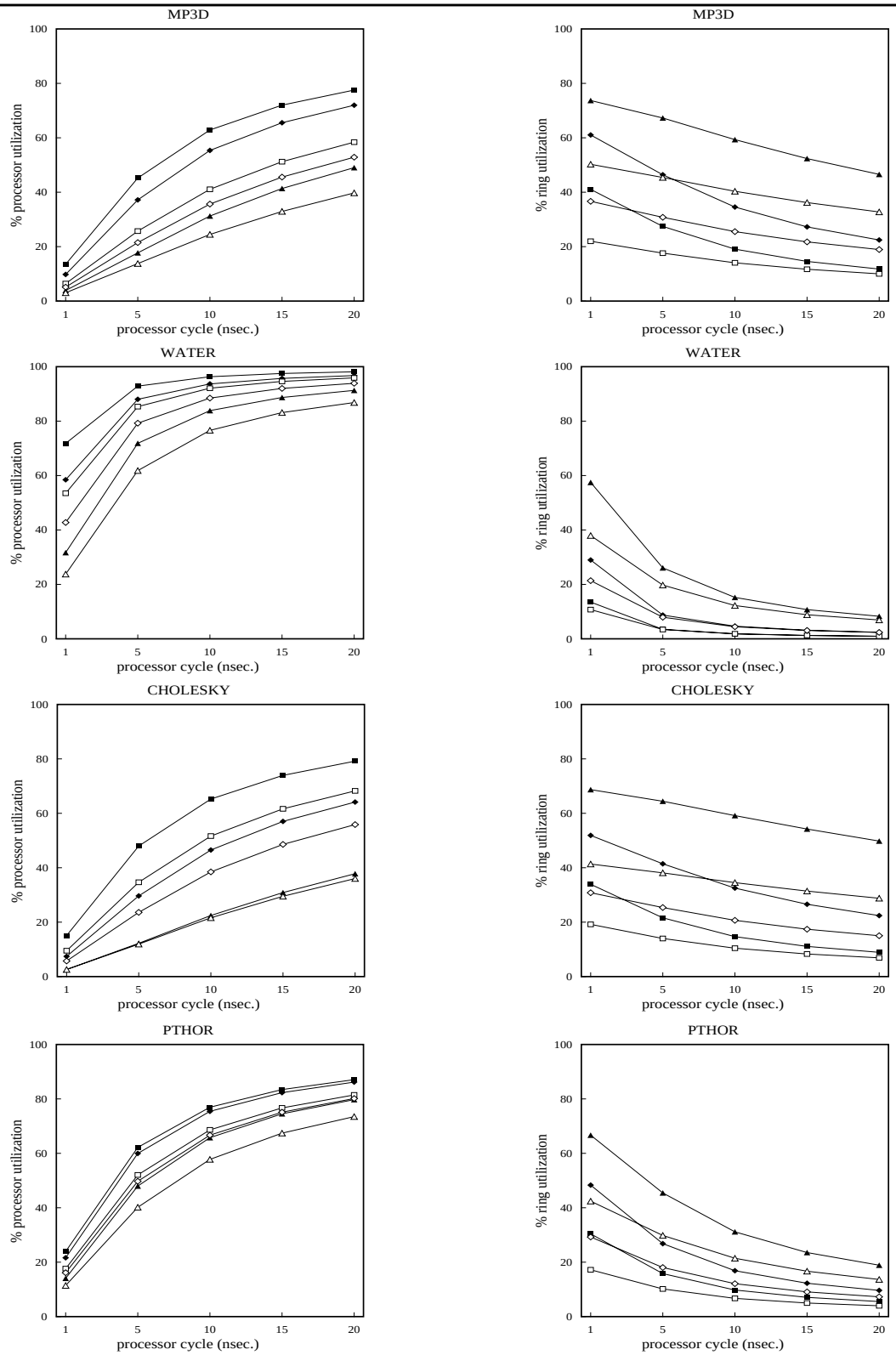
FIGURE 4. Breakdown of misses to shared data for the directory protocol (from the simulations)



The snooping protocol outperforms the directory protocol for all system sizes for MP3D because the fraction of 1-cycle dirty and 2-cycle misses is significant in all cases. The performance gap between the two schemes is not as wide for the 32 processor system, in which the fraction of remote clean misses is much larger.

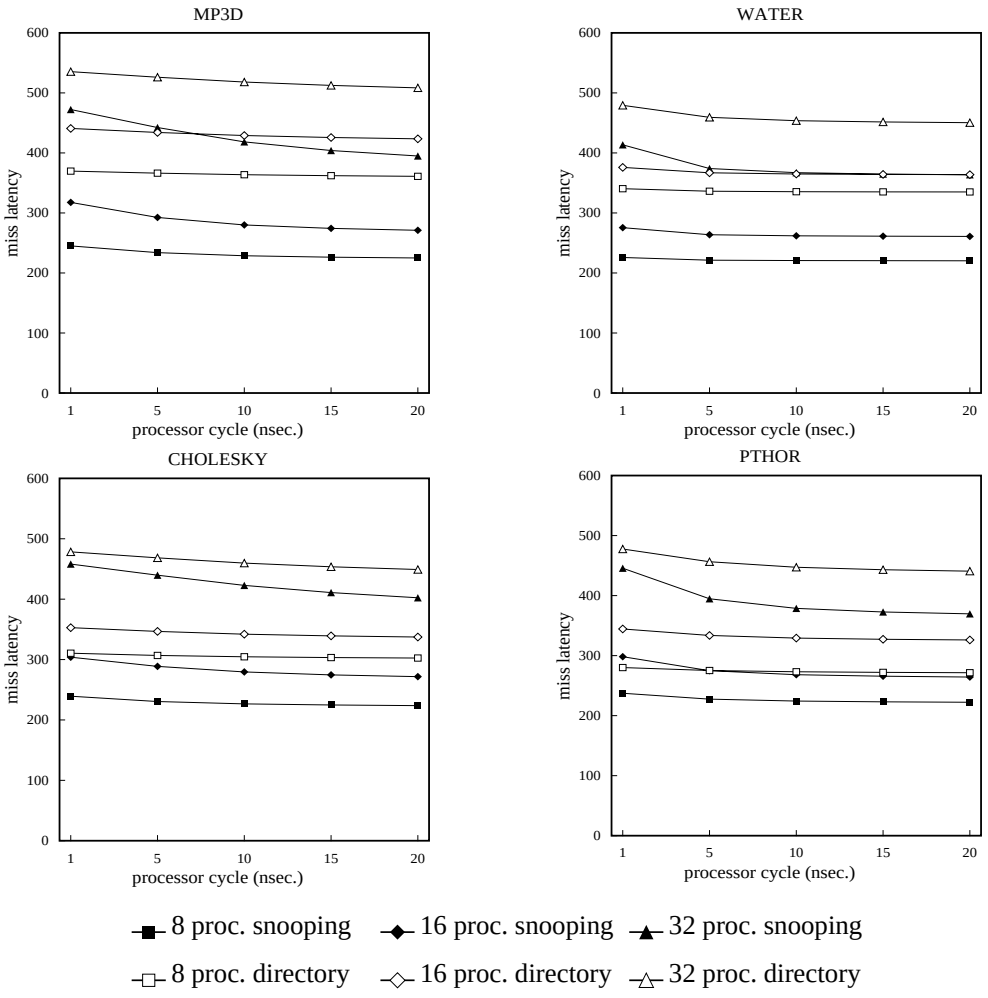
The ring utilization levels are always higher for snooping, as expected. However, as shown in Figure 6, the difference between the latencies of both protocols only narrows for the 32-processor MP3D. Two factors contribute to this: the increase in traffic starts to affect the latencies of snooping as the processor cycle decreases (the ring utilization of snooping is over 60% for processor speeds over 100 MIPS) and the larger fraction of remote clean misses in the directory protocol for the 32-processor case reduces the average miss latency.

FIGURE 5. Processor and ring utilization for snooping and directory (SPLASH)



For WATER, the extremely high hit ratio hides most of the differences between the snooping and directory protocols in terms of processor and ring utilization levels. The miss latency values however indicate the impact of the longer latency of 1-cycle dirty and 2-cycle misses. For the 8 and 32 processor cases, snooping starts to show a significantly better performance as the processor cycle decreases. CHOLESKY has a smaller fraction of 1 and 2-cycle misses for each system size than WATER and MP3D, and the difference between the latencies of misses for the two protocols is not as wide. For the 32-processor CHOLESKY, the miss latencies in the snooping systems are affected by contention delays and the processor utilization of the two schemes become roughly the same as the processor cycle decreases.

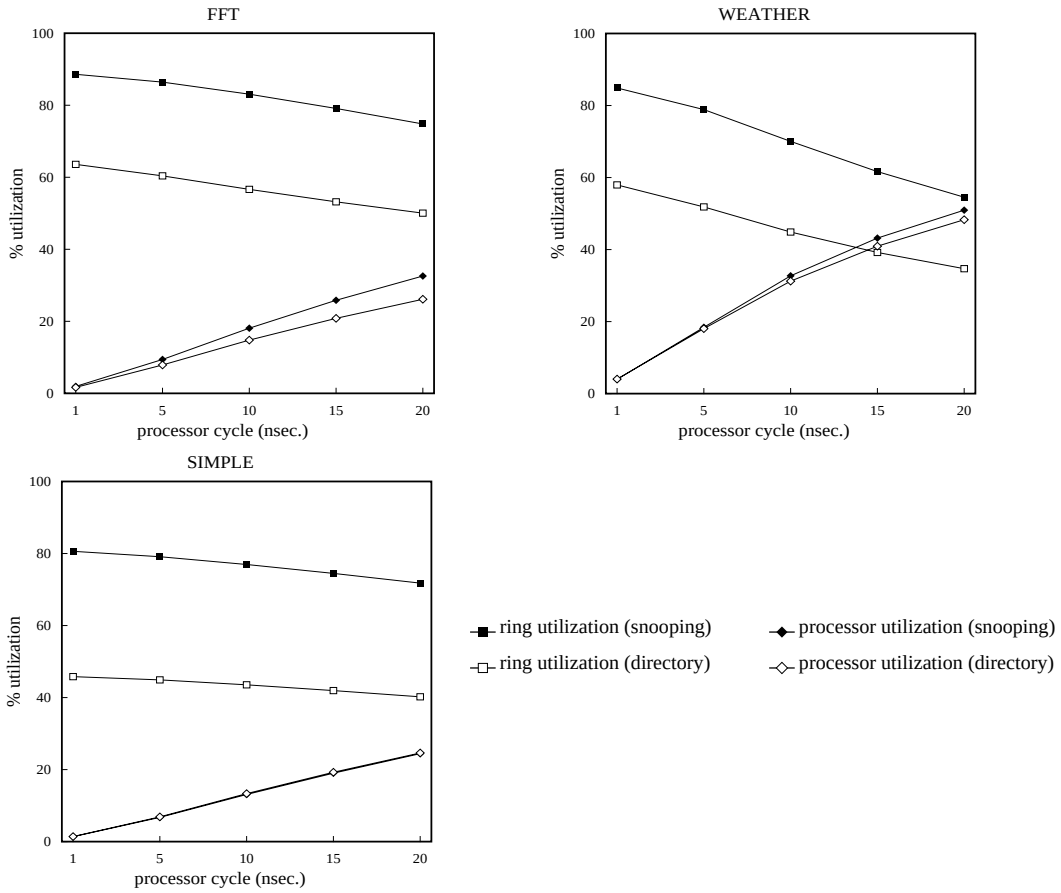
FIGURE 6. Miss latencies; snooping vs. directory



In PTHOR, even the 8 processor system has as a relatively small fraction of longer latency misses. However there is still a notable performance advantage for snooping for all cases in terms of processor utilization. Again, when the load in the interconnection starts to increase, the snooping protocol shows the effects of contention delays earlier than the directory protocol.

For FFT, SIMPLE and WEATHER, which are 64 processor traces, the processor utilization values drop considerably as a result of longer latencies. Again, the correlation between the mix of remote misses and the differences in performance between the two protocols is obvious. Among the three benchmarks, FFT is the only one that shows a significant number of 2-cycle misses and 1-cycle dirty misses. Consequently the snooping protocol shows a better average miss latency than the directory protocol for this trace when ring utilization values are relatively low. However, for SIMPLE, in which there is a very small fraction of higher latency misses, the difference in average latency figures is negligible. Once more, as the processor cycle decreases, the latencies of snooping surpass those of the directory protocol, due to contention delays.

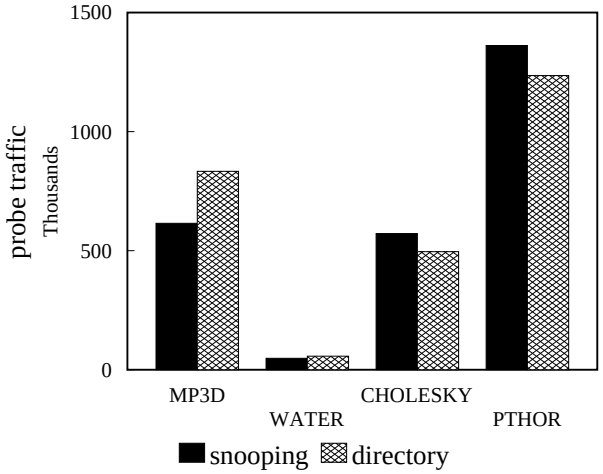
FIGURE 7. Processor and ring utilization for snooping and directory (MIT traces)



The general trend in the above evaluation is that whenever the ring utilization levels fall below 60%, the miss latencies are all but unaffected by contention. When the ring traffic increases, the contention delays affect snooping earlier than directory and the latency curves start to converge. Our simulation experiments with a 64-bit parallel slotted ring (not shown here) seem to agree with this assessment. With 64-bit parallel rings, utilization levels never surpass 50% and consequently, snooping performs far better than directory in all cases.

We have also observed that although the ring utilization values for snooping are always higher than for directory it is not true that the snooping scheme always generates more traffic.We have measured the message traffic in our trace-driven simulations as being the summation of all messages generated in one run, weighted by the fraction of the ring traversed by each message. The block message traffic is roughly the same for both schemes in all benchmarks. However, the probe traffic for the directory protocol is sensitive to the mix of remote misses, i.e., it tends to grow with the fraction of 1-cycle dirty and 2-cycle misses. In Figure 8 we show the probe traffic for 16 processor systems and a block size of 16 Bytes. We can see that for MP3D and WATER the probe traffic of snooping is actually lower than that of directory. This effect can not be seen in the ring utilization curves since the average ring utilization is measured over the execution time of the program, and the execution times for snooping are shorter. In fact, the main cause for the lower ring utilization values for the directory scheme is not lower traffic, but longer latencies and consequently longer execution times.

FIGURE 8. Probe traffic for 16 processor systems (simulation)



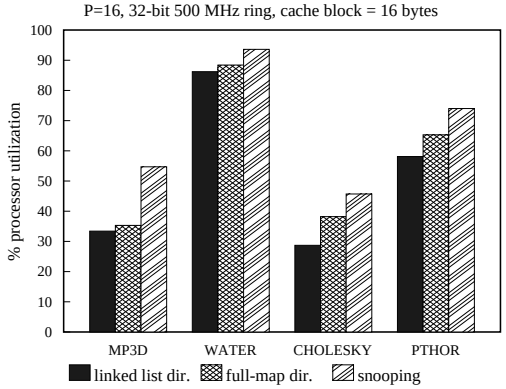
Finally, the snooping implementation requires faster logic in the ring interface and may incur higher costs than the directory protocol. Therefore, in design regions where the performance gap is not significant, a directory implementation may be preferred. The cost of snooping is mainly influenced by the minimum inter-arrival time of probes for a given dual-directory bank which depends on the ring width, clock cycle, and cache block size (see Table 2).

5.1.2 Linked list protocol performance

We now revisit our decision to use a full-map directory based protocol instead of a linked list protocol in the previous performance comparisons between snooping and directory protocols for the slotted ring. Figure 9 below shows the processor utilization for a 16 processor slotted ring with

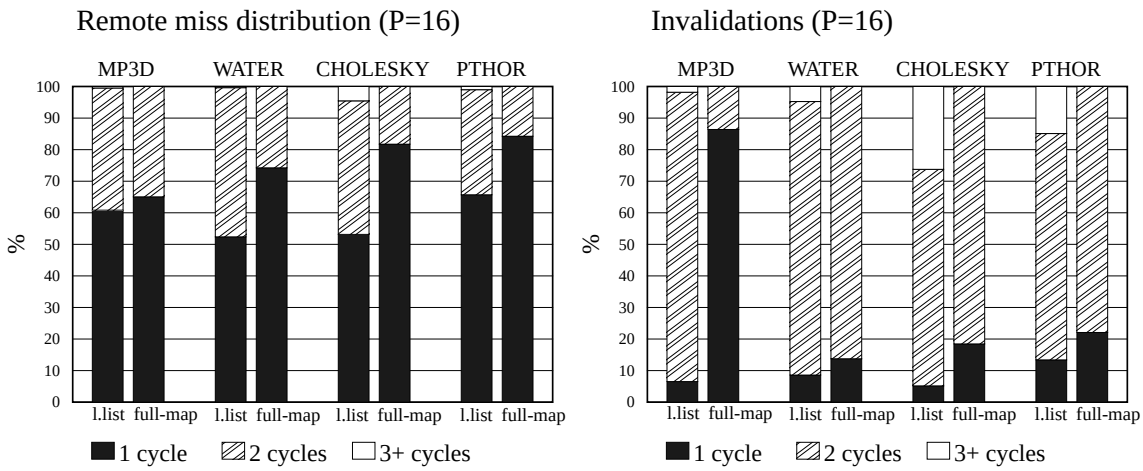
16 byte blocks and 100 MHz processors. The linked list protocol used is a version of the basic SCI coherence protocol [12] for a slotted ring. The logical behavior of the protocol is unchanged.

FIGURE 9. Processor utilization figures for a linked list directory protocol



We observed that, for all the SPLASH benchmarks used in this study, a linked list cache coherence protocol showed a slightly worse performance than a full-map directory protocol implementation. That is a result of the fact that in the linked list protocol a larger fraction of remote misses and invalidations require the ring to be traversed more than once, as shown in Figure 10. This agrees with the intuitive arguments presented in Section 3.3. One should note that these results are only valid for a unidirectional slotted ring multiprocessor, in which an extra hop in a cache coherence transaction has a good chance of adding one ring round trip delay to the total transaction latency.

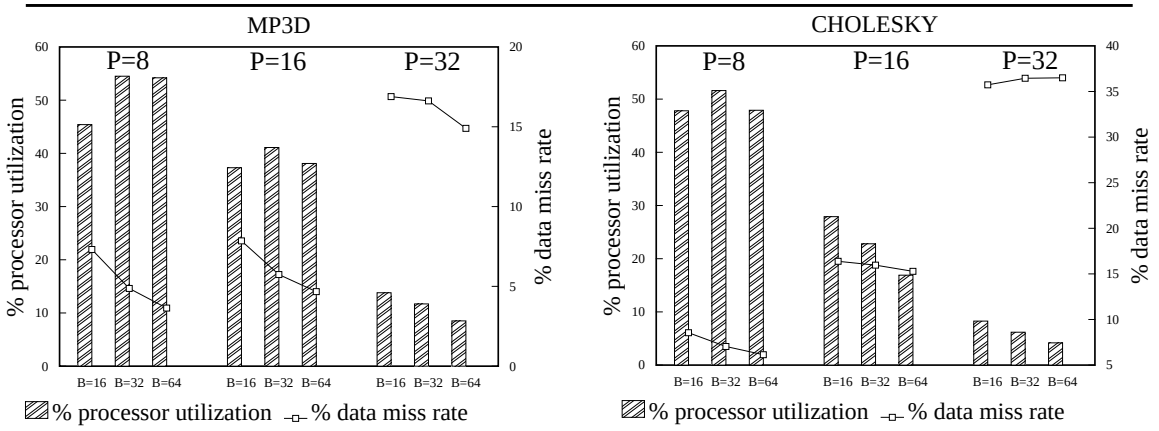
FIGURE 10. Breakdown of remote misses and invalidations (linked list and full-map dir. protocols)



5.1.3 Effect of the cache block size

To investigate the effects of varying the block size we have used trace-driven simulations only (no analytical model). We show the processor utilization results for snooping only, since the results for directory are quite similar. In Figure 11, the vertical bars indicate the processor utilization for systems with 8, 16 and 32 processors, with block sizes varying from 16 to 64 bytes. The corresponding miss ratios are shown as solid lines. The data cache size is fixed at 128 KBytes.

FIGURE 11. Effect of block size



If we use the product of the cache block size by the data miss rate as a rough approximation of the traffic, we can say that whenever the miss rate does not drop by a factor of two when we double the block size, the traffic in the ring will increase. The processor utilization factor is primarily influenced by the miss rate but it is also affected by the ring utilization as it translates into longer latencies due to contention delays. Another secondary effect of increasing the block size is that it decreases the number of message slots in the ring, therefore decreasing the parallelism in the interconnection. In general, whenever the increase in block size causes a significant decrease in the miss ratio and the ring utilization values are still low, the performance increases. This is the case for MP3D with P=8 and P=16 as the block size increases from 16 to 32 bytes, and also for CHOLESKY with P=8 as the block size increases from 16 to 32 bytes. When the larger block size does not lower the miss ratio enough (CHOLESKY, P=16), or when the traffic in the system was already high (MP3D and CHOLESKY with P=32), the performance will drop as the block size increases.

5.2 Slotted Ring vs. Split-Transaction Bus

We now use the snooping protocol to compare the performance of the slotted ring with that of a split-transaction shared bus. The bus architecture is similar to a split-transaction version of the FutureBus+ (IEEE 896.x standard), with a 3-state write-invalidate snoopy protocol and physical

shared memory partitioned among the processing nodes. Figure 12 compares the performance of a 32-bit wide ring, clocked at 500 MHz, to a 64-bit wide bus, clocked at 50 MHz and 100 MHz. The bus clock rates were chosen to represent aggressive values considering current technology and the range of system sizes.

The bus clock cycle remains constant across system sizes, which is somewhat optimistic because of the electrical limitations of buses mentioned previously. As a result, the pure latency to satisfy a remote miss remains constant for the bus case (assuming no contention), while it increases linearly with the number of nodes for the ring case. Using a 16-byte cache block, the minimum number of bus cycles to satisfy a remote miss is 6, excluding arbitration delays and the time to fetch the block in the remote node's memory or cache.

The limited bandwidth of the bus makes the actual miss latency values quite sensitive to variations in the processor speed, whereas the latency values for the ring remain nearly constant. Note that the processor speed is only one of the factors affecting the load on the interconnect. The average miss ratio for shared data and the fraction of shared data references are also good indicators of the load on the interconnect, for a given system size. Figure 13 displays the average bus utilization levels corresponding to the results in Figure 12.

MP3D (see Table 4) has a relatively high miss ratio for shared data and also has a significant fraction of shared data accesses. In the 8 processor MP3D the performance of the 100 MHz bus is comparable to the 500 MHz ring for slower processors ($\leq$ 50 MIPS), but it falls behind for increasingly faster processors due to bus conflicts. For the 16 processor MP3D the performance gap (in processor utilization) between ring and bus configurations increases as the buses enter saturation. In the ring configurations the network utilization is still under 50% even for 100 MIPS processors. In the 32 processor MP3D both buses are completely saturated, whereas the ring utilization stays under 80%. The behavior of CHOLESKY is very similar to MP3D.

The evaluations using WATER show a different behavior. In this case the miss rate values are extremely low, as is the fraction of references to shared data. The load on the interconnect is much lower than in MP3D. For P=8 and P=16, the bus starts to saturate for processor speeds higher than 200 MIPS. Even for 32 processors, the bus systems still show a very good performance level with 100 MIPS processors. For the 16 and 32 processor configurations, the pure latency of the 100 MHz bus is smaller than that of the 500 MHz ring. Therefore, for slower processors the bus configurations could outperform the slotted rings in the case of WATER, even if only by a narrow margin. However, in all cases, the slotted ring is less affected by contention

delays which is a result of its higher bandwidth. Eventually, as the buses reach saturation, the ring configurations have far better performance.

FIGURE 12. 32-bit slotted ring (500 MHz) vs. 64-bit split-transaction bus (100 MHz and 50 MHz)

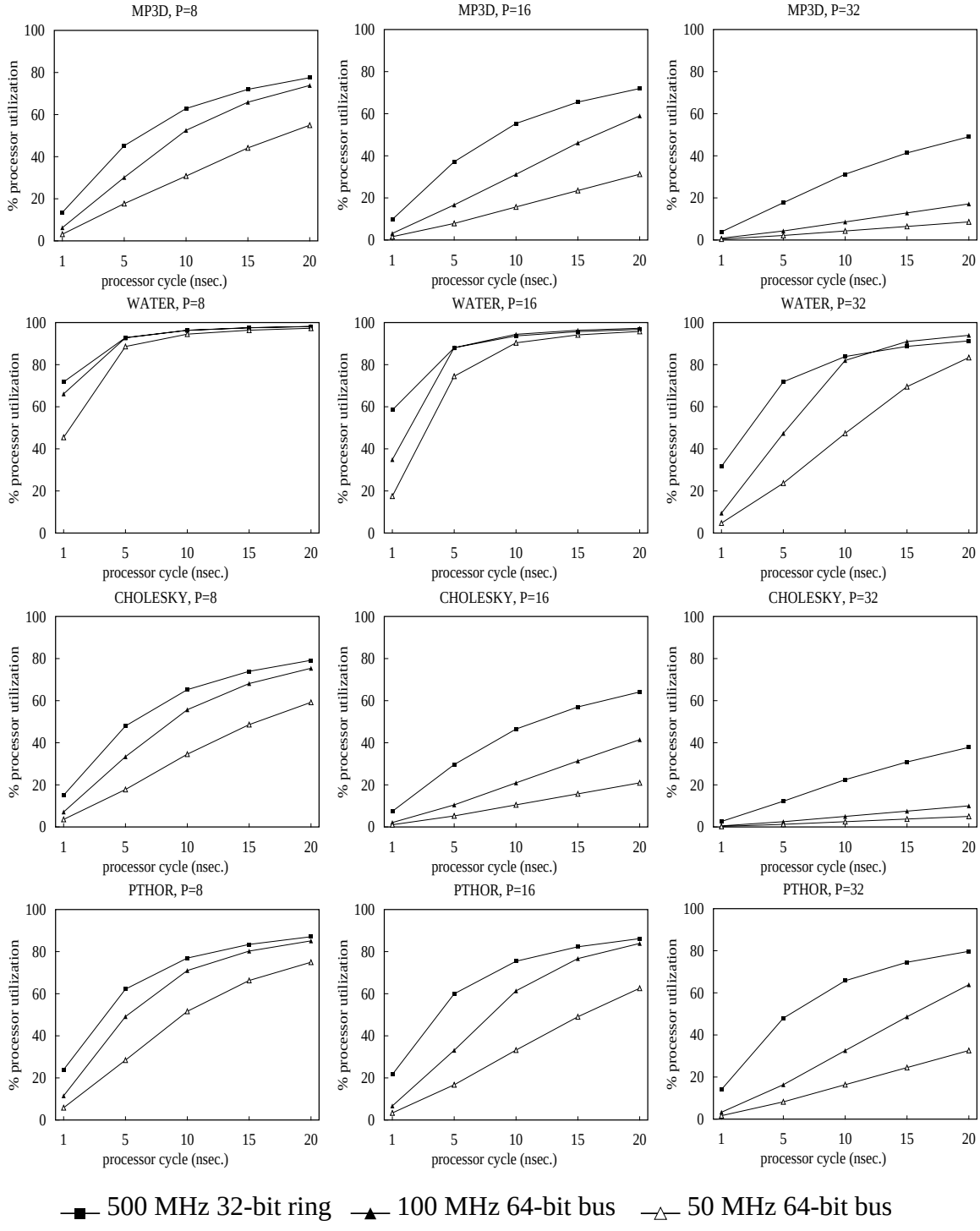
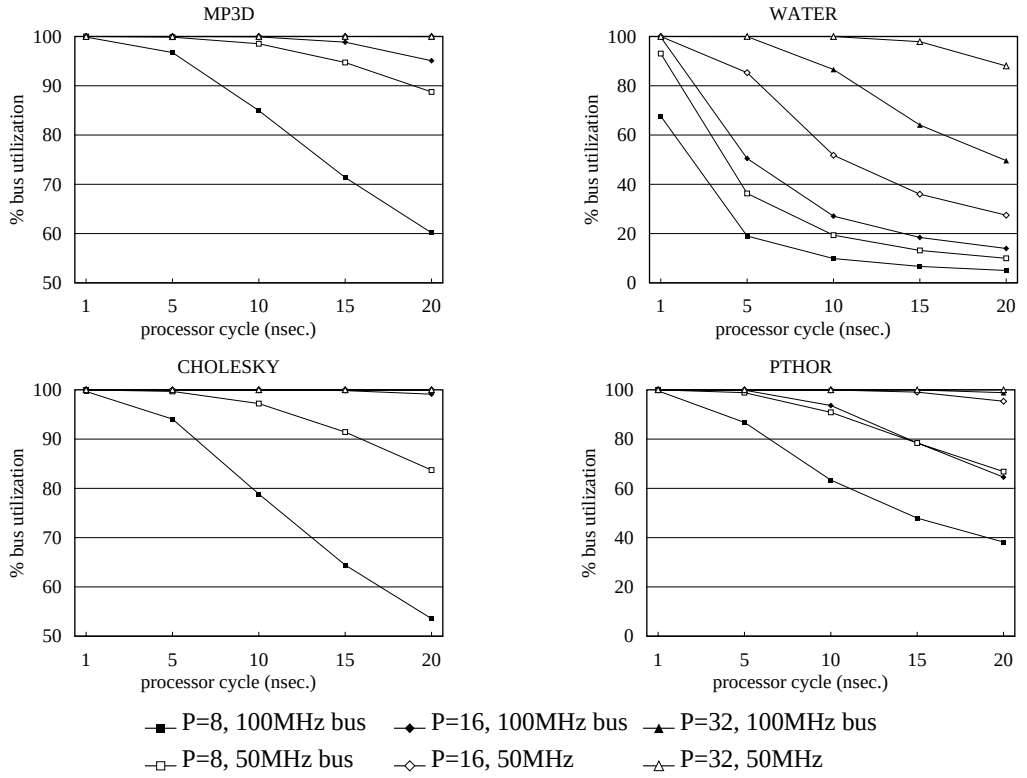


FIGURE 13. Bus utilization values; 64-bit split-transaction buses, 100 MHz and 50 MHz



In the case of PTHOR, the 100 MHz bus shows approximately the same processor utilization figures as the 500 MHz bus for systems with processing elements slower than 50 MIPS, and $P \leq 16$. As with the previous programs, as the processor cycle decreases the slotted ring outperforms the 100 MHz bus by up to a factor of three. For P=32, the performance gap between the slotted ring and the split-transaction bus increases even further, as the slotted ring is able to maintain reasonable processor utilization levels, but the buses enter saturation.

For 64 processors the bus systems are completely saturated in all cases. It is also unlikely that buses for 64 processors can be clocked at 50 MHz, using current bus technology. Therefore, we will not compare buses and rings for 64 processor systems.

The evaluation results showed here also indicate that the slotted ring could benefit from latency tolerance techniques, such as lockup-free caches, weak ordering schemes [9,11] and multithreaded processors [15] because the large latencies observed for the slotted ring are, in most cases, not caused by heavy contention but by pure delays. In other words, there is latency to be tolerated despite the fact that the network is often underutilized. Since most latency tolerance techniques have the collateral effect of increasing the load on the interconnect because of the overlap of communication and computation, they can be self-defeating in an interconnect working

close to saturation. This would probably happen in a split-transaction bus using very fast microprocessors. The latencies for the slotted ring however are still relatively stable and the network never saturates in the configurations that we have simulated. This indicates that the ring will be able to accommodate the increase in the load without significantly altering the expected latencies.

6.0 Related Work

The performance of unidirectional ring interconnections has been the subject of extensive analysis in the context of Local Area Networks (LANs) since the mid 70's. Bhuyan et al. [4] among several others have modeled the performance of ring LANs.

Our analytical models are based on Menasce and Barroso [14] but they are also very similar to the models used by Yang et al. [20] in their analysis of coherence protocols for bus-based multiprocessors. However, contrary to what is done in Yang et al. [20], we do not model the cache states of the protocols. Instead we take parameters from the trace-driven simulation of actual executions of parallel programs which simulate the caches and protocols in details.

In the context of distributed systems, Delp et al proposed a token ring distributed shared memory system (called Memnet [7]) with cache coherence maintained in hardware by means of a snooping-like coherence protocol. More recently Scott et al [16] have analyzed the performance of the SCI ring, which is an implementation of the register insertion access control strategy. They model the ring as a M/G/1 queue and derive the expected latency of messages with respect to network throughput, assuming an exponentially distribution for arrival times of messages. The authors point out that fairness of access is a problem in the register insertion approach, and it may lead to the starvation of a node. The mechanism proposed by SCI to avoid starvation is shown to impact the effective throughput of the ring. In the slotted ring it is very simple to avoid starvation of clusters by not allowing a node to reuse a message slot immediately after removing a message from that slot. Our simulations show that this has no significant impact on the system performance. Scott also makes a comparison between the register insertion ring and a shared bus, but without taking the cache coherence level into consideration.

It is not clear for us which scheme, slotted or register insertion, offers the best performance. Intuitively, under very light loads the register insertion ring may have faster access since a message does not wait for a proper slot to pass by. On the other hand, under medium to heavy loads, the simplicity of enforcing fairness on the slotted ring may yield a better performance than the register insertion approach. The delay of transmitting a message in the register insertion ring can vary significantly depending on the activity of other nodes in the

message path. We also believe that it is simpler to implement a slotted ring interface because it does not require a bypass buffer.

The Hector multiprocessor [19], at the University of Toronto, uses unidirectional token rings similar to our slotted ring. It is a shared memory multiprocessor with a three level interconnection hierarchy. Processing elements with a private cache and a part of the physical memory space are connected to a shared bus forming what are called *stations*. Stations are linked together by a *local ring*, local rings are connected by a second level *global ring*. All rings are unidirectional. Even though the initial Hector architecture did not include hardware support for cache coherence (shared data were marked as uncachable), more recent work by Farkas at al. [10] recognizes the need for it and specifies a hierarchical cache protocol based upon the broadcasting of requests, as in the snooping protocol used here.

The only commercial implementation of a ring connected multiprocessor that we are aware of at this time is the Kendall Square Research KSR1 [13]. It is a shared memory cache-coherent multiprocessor based on a two-level hierarchy of unidirectional rings that uses a memory strategy called *Allcache*. In this strategy, the usual main memory is replaced by a very large cache. We are unable to compare the snooping protocol studied here with KSR1's first level snooping protocol due to the absence of technical information on it.

7.0 Conclusion

In this paper we have evaluated a particular interconnection for small to medium scale shared memory multiprocessor systems: the unidirectional slotted ring. A comparative analysis of two cache coherence strategies, snooping and directories, for the slotted ring was performed with a hybrid methodology incorporating detailed trace-driven simulations and analytical models. We have determined that a full-map directory protocol outperforms a linked list directory protocol in the context of the slotted ring. That is because the linked list protocol has a larger fraction of coherence transactions that require multiple ring traversals.

We thoroughly compared the performance of the snooping cache coherence protocol with that of the full-map directory protocol. Contrary to common wisdom, the snooping strategy outperforms the directory strategy for all system configurations analyzed. In some cases, snooping may even generate less traffic than the directory protocol. We justified the differences in performance between the two protocols by looking into the particular sharing patterns of the various benchmarks used in the evaluations. We also compared the performance of the slotted ring under snooping with that of a split-transaction bus connected multiprocessor. The results show that even for systems with as few as 8 processors, the slotted ring can outperform a

pipelined split-transaction bus, for benchmarks with a significant fraction of remote misses and invalidations. Whereas the speed of bus interconnections are expected to lag further and further behind with respect to microprocessors, the ring interconnection is likely to keep up with future generations of microprocessors.

8.0 Acknowledgments

We would like to thank Per Stenström and his research group for providing us with the CacheMire Test Bench which we used to derive the traces for this study.

9.0 References

[1] L. Barroso and M. Dubois, “Cache Coherence on a Slotted Ring”, Proceedings of the 1991 International Conference on Parallel Processing, Vol. I, pp. I230-I237 , St. Charles, IL, August 1991.

[2] L. Barroso and M. Dubois, “The Performance of Cache-Coherent Ring-based Multiprocessors”, Proceedings of the 20th International Symposium on Computer Architecture, pp. 268-277, San Diego, CA, May 1993..

[3] M. Brorsson, F. Dahlgren, H. Nilsson and P. Stenström, “The CacheMire Test Bench - A Flexible and Efficient Approach for Simulation of Multiprocessors”, Proceedings of the 26th Annual Simulation Symposium, March 1993.

[4] L. Bhuyan, D. Ghosal, and Q. Yang, “Approximate Analysis of Single and Multiple Ring Networks”, IEEE Transactions on Computers, Vo. 38, No. 7, pp. 1027-1040, July 1989.

[5] L. Censier, and P. Feautrier, “A New Solution to Coherence Problems in Multicache Systems”, IEEE Transactions on Computers, C-27(12), pp. 1112-1118, December 1978.

[6] D. Chaiken, C. Fields, K. Kurihara and A. Agarwal, “Directory-Based Cache Coherence in Large Scale Multiprocessors”, IEEE Computer, Vol. 23, No. 6, pp. 49-59, June 1990.

[7] G. Delp, D. Farber, R. Minnich, J. Smith and M-C. Tam, “Memory as a Network Abstraction”, IEEE Network Magazine, pp. 34-41, July 1991.

[8] Digital Equipment Corp., “Alpha Architecture Handbook”, DEC, Massachussets, February 1992.

[9] M. Dubois and C. Scheurich, “Memory Access Dependencies in Shared Memory Multiprocessors”, IEEE Trans. on Software Engineering, 16(6), pp. 660-674, June 1990.

[10] K. Farkas, Z. Vranesic and M. Stumm, “Cache Consistency in Hierarchical Ring-Based Multiprocessors”, Proceedings of Supercomputing’92, November 1992.

[11] A. Gupta, J. Hennessy, K. Gharachorloo, T. Mowry and W.D. Weber, “Comparative Evaluation of Latency Reducing and Tolerating Techniques”, Proceedings of the 18th International Symposium on Computer Architecture, pp. 254-263, Toronto, Canada, May 1991.

[12] D. Gustavson, “The Scalable Coherent Interface and Related Standards Projects”, IEEE Micro, Vol. 12, No. 1, pp. 10-22, February 1992.

[13] Kendall Square Research, “Technical Summary”, Walthan, Massachusetts, 1992.

[14] D. Menasce, and L. Barroso, “A Methodology for Performance Evaluation of Parallel Applications in Multiprocessors”, Journal of Parallel and Distributed Computing, Vol 14, No. 1, pp. 1-14, January 1992.

[15] R. Saavedra-Barrera, D. Culler and T. von Eicken, “Analysis of Multithreaded Architecture for Parallel Computing”, 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 169-178, Greece, July 1990.

[16] S. Scott, J. Goodman and M. Vernon, “Performance of the SCI Ring”, Proceedings of the 19th International Symposium on Computer Architecture, pp. 403-414, Gold Coast, Australia, June 1992.

[17] H. Schwetman, “CSIM: A C-Based, Process-Oriented Simulation Language”, Proceedings of the 1986 Winter Simulation Conference, pp. 387-396, 1986.

[18] J. Singh, W-D. Weber and A. Gupta, “SPLASH: Stanford Parallel Applications for Shared Memory”, SIGArch Computer Architecture News, Vol. 20, No. 1, pp. 5-43, March 1992.

[19] Z. Vranesic, M. Stumm, D. Lewis and R. White, “Hector: A Hierarchically Structured Shared Memory Multiprocessor”, IEEE Computer, Vol. 24, No. 1, pp. 72-78, January 1991.

[20] Q. Yang, L.N. Bhuyan and B.-C. Liu, “Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor”, IEEE Transactions on Computers, Vol 38, No. 8, pp. 1143-1153, August 1989.