The Journal of Supercomputing, 8, 163–185 (1994)

A Cache Coherence Protocol for MIN-based Multiprocessors

MAZIN S. YOUSIF (*yousif@engr.LaTech.edu*)
Department of Computer Science, Louisiana Tech University, P.O. Box 10348, Ruston, LA 71272

CHITA R. DAS (*das@cse.psu.edu*)
Department of Computer Science & Engineering, The Pennsylvania State University, University Park, PA 16802

MATTHEW J. THAZHUTHAVEETIL (*mjt@csa.iisc.ernet.in*)
Supercomputer Education and Research Center and Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India

(Received June 1992; final version accepted February 1994.)

Abstract. In this paper we present a cache coherence protocol for *multistage interconnection network* (MIN)-based multiprocessors with two distinct private caches: *private-blocks caches* (PCache) containing blocks private to a process and *shared-blocks caches* (SCache) containing data accessible by all processes. The architecture is extended by a coherence control bus connecting all shared-block cache controllers. Timing problems due to variable transit delays through the MIN are dealt with by introducing *Transient* states in the proposed cache coherence protocol. The impact of the coherence protocol on system performance is evaluated through a performance study of three phases. Assuming homogeneity of all nodes, a single-node queuing model (phase 3) is developed to analyze system performance. This model is solved for processor and coherence bus utilizations using the mean value analysis (MVA) technique with shared-blocks steady state probabilities (phase 1) and communication delays (phase 2) as input parameters. The performance of our system is compared to that of a system with an equivalent-sized unified cache and with a multiprocessor implementing a directory-based coherence protocol. System performance measures are verified through simulation.

Keywords. Caches, cache coherence, mean value analysis, multiprocessor system, multistage interconnection network, split cache.

1. Introduction

Maintaining coherence of shared data in multiprocessors with private cache memories is essential for the correct execution of a program. In view of this, a number of cache coherence protocols for shared-memory multiprocessors have been proposed in recent years [Eggers 1989]. Most of the proposed protocols assume a bus-based multiprocessor as the underlying architecture. They rely on *broadcasting* coherence maintenance signals on the bus for keeping cache contents consistent. Although a variety of MINs have been proposed for building large multiprocessors, less effort has been directed at developing coherence schemes for such systems, primarily because of the toll of broadcasting in terms of time overhead and the lack of *order-of-requests* within a MIN.

A few cache coherence protocols that require an extension of the basic MIN architecture have recently been proposed [Bhuyan et al. 1989; Mizrahi et al. 1989; Stenström 1989; Yousif 1991]. Extensions are designed to allow for the quick sharing of information to maintain cache coherence. A snoopy bus can be added to connect all cache controllers [Bhuyan et al. 1989; Yousif 1991]. The sequentiality of operations inherited in a bus is exploited to maintain consistency of the shared data in caches. In [Mizrahi et al. 1989] small memories containing global directory information are added to the switching elements of a MIN. Here, data inconsistency is avoided by allowing *only one* copy of the shared data outside of main memory. A third approach, suggested by Stenström [1989], is based on a distributed directory (added to each cache controller) and a central directory at main memory. (A number of cache coherence schemes for network-based multiprocessors were proposed earlier [Censier and Feautrier 1978; Tang 1976; Yen and Fu 1982]. A few software-assisted schemes have also been proposed [Cheong and Veidenbaum 1988; Min and Baer 1989; Smith 1985]. Recently, there have been attempts to build large-scale cache-coherent multiprocessors, such as the Stanford DASH [Lenoski et al. 1990], the MIT Alewife [Agarwal et al. 1990], and the Wisconsin Multicube [Goodman and Woest 1988]. The IEEE standard project P1596 is an attempt to design a backplane interface referred to as the *scalable coherent interface* (SCI) [Gustavson 1992].

In the schemes that complement the base architecture with a coherence control bus, it was observed that the coherence bus is still the classic bottleneck that limits the scalability of such approaches [Bhuyan et al. 1989; Yousif 1991]. The amount of traffic activity on the bus with the limited bus bandwidth reflects the number of nodes that can be sustained by a bus. In this paper we will improve the scalability of this approach by further reducing traffic activity on the bus. This traffic reduction is obtained by splitting the cache into two distinct units, as explained later. A cache coherence protocol is presented; then its impact on system performance is evaluated. The main contribution of this paper is the proposed architecture/cache coherence scheme and the system performance evaluation methodology.

Based on the accessibility of a cache block, it is possible to split a local cache into two distinct parts: a PCache, which contains blocks that are *private* to a process, and an SCache, which contains blocks that are *shared* among processes. To implement this, we assume that private and shared blocks are distinguishable. (This might be done, for example, by associating a separate address range for each.) We introduce the concept of accessibility-based split caches to take advantage of the inherent characteristics of reference streams. References to private data (e.g., instructions) do not pose memory coherence problems, since private blocks cannot become inconsistent. Therefore, PCaches add no overhead other than that due to a private-block miss. However, in addition to the overhead incurred due to an SCache miss, in-SCache shared data modification leads to the cache coherence problem. The motivation of this approach is to eliminate the effect of private blocks on cache coherence that is present in a unified cache, as will be discussed later.

In this paper an accessibility-based split-cache coherence protocol for MIN-based multiprocessors is proposed and evaluated. To implement the protocol, the architecture is extended by adding a *coherence control* (snoopy) bus that connects all SCache controllers. Block transfers to and from the shared memory are through the MIN. Adding such a coherence bus makes it possible to implement coherence protocols designed for bus-based multiprocessors on a MIN-based architecture with minimal protocol modification. (This

approach can be extended to any interconnection network-based multiprocessor.) Since we choose a write-update protocol for this paper, the bus is used for write broadcasts—cache-to-cache block transfers to update cached shared-block copies upon data modification. SCache controllers snoop at all bus transactions.

The basic architecture under consideration is a shared-memory multiprocessor with N processor nodes connected through a MIN, as shown in Figure 1. Each processor node includes a local memory that is globally accessible. In other words, the shared memory is distributed among the nodes of the multiprocessor, as in the BBN Butterfly [BBN 1989]. The Butterfly MIN is adopted as the base MIN for this study. Each node also includes the following: two distinct caches, a PCache and SCache; a local directory; and a network interface plus other required control hardware. SCache controllers are connected by a coherence control bus.

Based on the proposed protocol, we develop a comprehensive evaluation methodology for analyzing system performance. It includes three phases: a protocol's states probabilities computation, a MIN delay model, and a node queuing model. The protocol is first specified in terms of cache coherence operations and state transitions. We introduce *Transient* states to reflect the effect of the traffic-dependent variable delay through the MIN. Shared-blocks steady state probabilities of the states imposed by the protocol are then calculated—they play a role in determining the amount of traffic in the MIN. The communication delay through the MIN is computed using a queuing model of the MIN. Next, assuming that all nodes perform identical tasks, a queuing model for a single node is constructed. This model depicts the flow of a processor request in the system under coherence (whether private or shared, local or remote, hit or miss, etc.). It requires shared-blocks steady state probabilities and average communication delays as input parameters. The single-node queuing

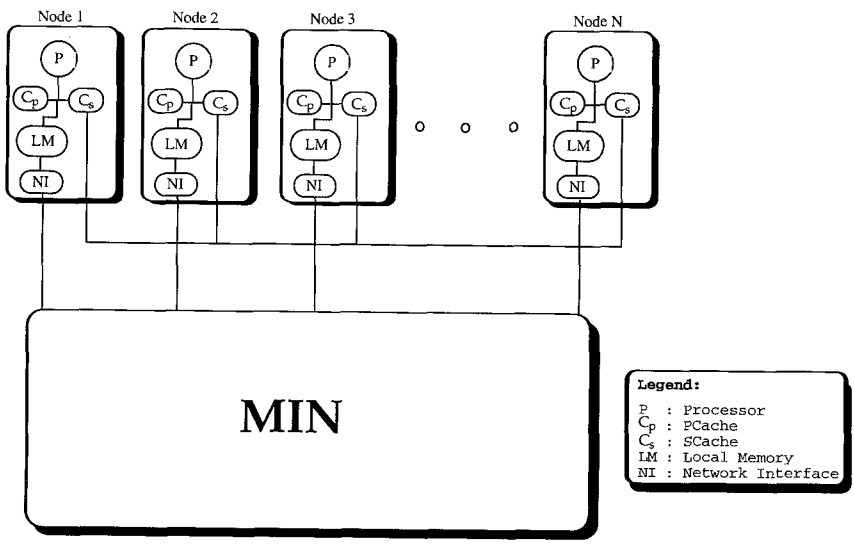

Figure 1. A multiprocessor system of size N.

model is then solved using the mean value analysis (MVA) technique for finding system performance measures such as processor utilization and coherence bus utilization [Lazowska et al. 1984].

This paper is organized as follows: Section 2 describes the proposed coherence protocol. In Section 3 the system workload model is presented. Section 4 presents an evaluation study to calculate the impact of the protocol on system performance. Section 5 concludes this work.

2. The Cache Coherence Protocol

The cache coherence protocol is a write-update protocol; that is, when a shared block in a cache is modified, copies of the block at other caches are updated [Smith 1982]. A local directory maintains a three-bit state entry for each shared data block in a cache. The bits designate a block as *Transient* or *Permanent*, *Modified* or *UnModified*, and *Exclusive* or *NonExclusive*. Transient states are introduced to reflect the delay of fetching/writing back a block from/to memory through the MIN. A remote or local memory access could take several clock cycles, depending on network traffic, location of the block, and system size. A Transient state indicates that the block is in transit through the MIN due to a block transfer request—the block cannot be accessed in cache until both the data arrive and the state changes to Permanent. A block in a Permanent state is available in cache. The bit designated Exclusive/NonExclusive shows whether the shared block is in only one or more than one cache. (If the Exclusive bit is set, this is the only cached copy.) The protocol guarantees that copies of a shared block in other caches are updated on block modification if the Exclusive bit is *not* set. (The update is carried over the coherence bus.)

State transitions for a shared-data block occur due to the following incoming requests, as shown in Figure 2: a read or write from the local processor, referred to as a *local read* and *local write*, respectively, and a read or write from one of the remaining $N - 1$ processor nodes that is propagated on the coherence bus, referred to as a *remote read* and *remote write*, respectively. Local reads/writes combine the following cases: read/write hits; read/write misses with no cached copies of the block in other caches; and read/write misses with cached copies of the block in other caches.

The state of a block changes depending on the type of request and the current state of the block, as specified below. Note that states are *Permanent* unless indicated otherwise.

1. *Read hit:* A read hit is satisfied locally without any state change.
2. *Read miss:* Initially, block replacement is done if necessary. The cache controller broadcasts a *block-check* request on the bus. One of the following cases is possible.

   Case 1: All remote cache controllers respond with negative acknowledgments. Therefore, the block is still in main memory. The block state is set to UnModified-Transient. A *block-fetch* request is sent by the cache controller through the MIN to the memory controller. When the block arrives at the cache, the block's state is updated to UnModified-Exclusive.

   Case 2: At least one remote cache responds with the state of the block, which could be either Transient or Permanent.

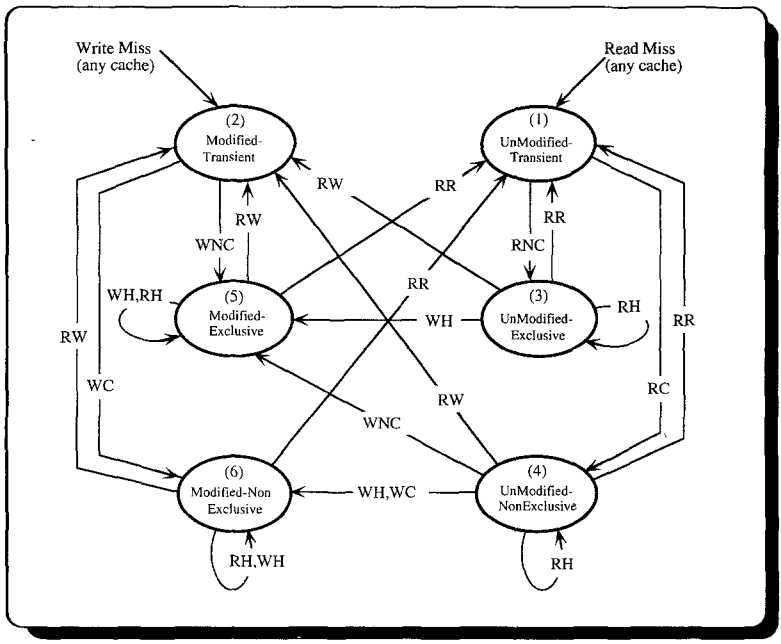

Figure 2. The protocol state transition diagram.

a. If the block is in a Transient state, the request is retried after a predetermined delay (this delay is a proportional average of the local and remote memory reference delays). The processing of the read miss can proceed only when the state has changed to Permanent, as described below.

b. If the state of the block is Permanent, the block could be either Modified or UnModified and Exclusive or NonExclusive. The block's state is changed to UnModified-Transient in all cases. For the UnModified (Exclusive or NonExclusive) case, the cache controller sends a block-fetch request through the MIN to the memory controller to fetch the block. Then, the state of the block is changed to UnModified-NonExclusive. (Note that this is a Permanent state.)

If the block's state is Modified-Exclusive, then in response to the block-check request, the cache controller with the updated copy will write back the block to main memory. To avoid timing problems, an acknowledgment signal is returned from the memory controller to this cache when the write-back is completed. After that the

cache broadcasts a signal on the bus informing other caches that the write-back has been completed. Then, a block-fetch request is sent from the cache controller (cache with the read miss) to the memory controller. The block's state is changed to UnModified-NonExclusive. (We prefer to write-back/fetch the updated copy of the block through the MIN rather than perform direct block transfer through the bus in order to reduce the traffic activity on the bus. This will improve the scalability of the system since the coherence bus is the likely bottleneck in this design.) For the Modified-NonExclusive case, the set of operations is similar to the Modified-Exclusive case above, except that when the block-check request is broadcast, the first cache with a copy of the block that responds will remove the request from the bus to prevent other caches with copies of the block from responding. The state of the block remains UnModified-NonExclusive.

3. *Write hit:* If the block is Transient, the write hit cannot proceed until the block becomes Permanent, as shown below. A write to a Modified-Exclusive block (i.e., the only copy) brings on no remote activity and the block's state remains Modified-Exclusive. However, a write to a Modified-NonExclusive block requires broadcasting the write through the bus to update copies of the block in other caches. A globally UnModified block could be either Exclusive or NonExclusive. A write to an UnModified-Exclusive block requires only updating the block's state to Modified-Exclusive. For an UnModified-NonExclusive block, a write broadcast through the bus is required to update copies of the block in other caches. Then the block's state is changed to Modified-NonExclusive.
4. *Write miss:* Initially, block replacement takes place if needed. The cache controller broadcasts a block-check request over the bus. As in the read miss, the following two cases are possible.

   Case 1: All remote caches respond with negative acknowledgments. Therefore, the block is still in main memory. The block's entry in the local directory is set to Modified-Transient. A block-fetch request is sent by the cache controller to the memory controller. After the arrival of the block, the cache controller updates the block's state to Modified-Exclusive.

   Case 2: At least one cache responds with the state of the block, which could be either Transient or Permanent.

   a. If the block is Transient, the write miss is delayed until the block becomes Permanent, as described below.
   b. If the block is Permanent, the block could be either Modified or UnModified, and Exclusive or NonExclusive. Initially, the block's state is set to Modified-Transient in all cases. For a globally UnModified (Exclusive or NonExclusive) block, a block-fetch request is sent by the cache controller to the memory controller. When the block arrives at the cache, a write-update request is broadcast over the bus to update copies of the block in other caches. The block's state is changed to Modified, either Exclusive or NonExclusive. When the block is globally Modified, operations similar to the read miss case are processed, except that the block's state is updated to Modified, either Exclusive or NonExclusive.
5. *Shared-block replacement:* When a Modified-Exclusive shared block is chosen for replacement due to a cache miss, the block's state is first changed to Modified-Transient. Then, a write-back is performed through the MIN to the relevant memory. After receiving

the acknowledgment that the write-back has been completed, the block's entry in the directory is removed. When a Modified-NonExclusive shared block is chosen for replacement, a write-back is performed through the MIN to memory. (A count check is now appropriate in order to find the remaining number of block copies. The cache controller broadcasts a *nonexclusive-exclusive* request over the bus. If more than one cache controller responds with a positive acknowledgment, the block's state is retained as Modified-NonExclusive. Otherwise, if only one cache controller responds, the block's state is updated to Modified-Exclusive.) If the block is UnModified-Exclusive, the block's entry in the local directory is deleted. But if the block is UnModified-NonExclusive, no action is taken. (A check for the remaining number of copies of the block might also be in place here, as mentioned above.)

We do not have a formal proof to verify the correctness of the coherence protocol. It is possible to conclude that this protocol does not have any timing problems by exhaustively checking the transitions among the states of the protocol in the transition diagram [Archibald 1987].

3. Workload Model

We assume that all processor nodes are identical—each processor possesses two distinct caches: SCache and PCache, of sizes ψ_{cs} and ψ_{cp} blocks, respectively. A processor "computes" for a certain period of time, then generates a request to one of its caches. The computation time of a processor is assumed to follow geometrical distribution with average value z. A geometrical distribution indicates that a processor issues requests only at discrete clock cycles.

The system workload model consists of two memory reference streams, one to private blocks and another to shared blocks. Dubois and Briggs [1982] used an *independent reference model* (IRM) for shared-block accesses and a *least recently used stack model* (LRUSM) for private block accesses. An IRM for shared-block accesses is not adopted here since it does not capture any locality. Since shared-block accesses possess some locality (less than that for private-block accesses), we will assume LRUSM models with different localities for both referencing schemes. Both request patterns are modeled as stacks that are unique to each processor. The contents of the stack reflect the past reference pattern of a processor, with the most recent reference at the top. The probability of requesting a block at depth j in a stack model is represented by

$$P[j] = G(M)[(l + j)^{-1} - (l + j + 1)^{-1}], \tag{1}$$

where $G(M)$ is a normalization factor that forces $\Sigma_{j=1}^{M} P[j] = 1$ [Archibald 1987]; M is the number of blocks in the stack and the parameter l reflects the temporal locality. In the simulations conducted for this study, we use $l = 5$ for shared-block references and $l = 3$ for private-block references. These values of l were chosen based on the observation in [Archibald 1987] that for uniprocessors, $l = 5$ results in a shared-block hit ratio comparable to a private-block hit ratio of 0.95. Since a smaller l models a greater degree of

temporal locality, we use $l = 3$ for private-block references, based on the assumption that shared blocks are referenced with less locality than private blocks.

The following notations are used in this paper.

- f_r Probability of a read operation.
- f_w Probability of a write operation; $f_r + f_w = 1$.
- h_{cp} PCache hit ratio.
- h_{cs} SCache hit ratio under coherence.
- md Probability that a private block is modified.
- N Number of nodes in the system (system size).
- N_{sh} Number of shared blocks in the system.
- q_p Probability of making a request to a private block.
- q_s Probability of making a request to a shared block (*degree of sharing*); $q_s + q_p = 1$.
- S_b Coherence bus service time.
- S_c Cache service time.
- S_g Average delay for a block fetch from a remote memory module through the MIN.
- S_l Average delay for a block fetch from the local memory module.
- S_{se} Switching element service time.
- U_b Probability that the coherence bus is busy (bus utilization).
- U_p Probability that a processor is busy (processor utilization).
- z Processor think time.
- λ_p Traffic rate generated by a processor.
- ψ_{cp} PCache size in blocks.
- ψ_{cs} SCache size in blocks.

4. System Performance

To study the impact of the coherence protocol on system performance, a performance study that includes both simulation and analysis is conducted. Extensive discrete-event simulations that are driven by stochastically generated traces are run. The workload model for these simulations was presented in Section 3. Each simulation represents an exact flow of processor requests through the system with cache coherence enforced.

A processor does internal computation for a certain period of time, z, before generating a request to one of its caches. (A private-block request is directed to the PCache and a shared-block request is directed to the SCache.) This request is either a cache hit or miss. If it is a cache hit, the processor resumes computation after a delay related to the access time of that cache; otherwise, either the request is a cache miss, or extra sevice such as broadcasting updates is required. A private-block cache miss is satisfied either locally or from a remote memory module through the MIN. If the request is a shared-block cache

miss, the cache controller broadcasts a request through the coherence bus to check the remaining caches for the possible presence of the block. If a remote cache has the block, it writes it back to main memory, which forwards it to the requesting cache. This process includes changing states between Transient and Permanent. For writes, the update is broadcast over the bus to update copies of the block in other caches, if any. Input parameters used in the simulation are similar to those adopted in the analysis, as listed in Table 1.

The analytical part of our performance study includes two parts: (1) a shared-blocks steady state probabilities computation and (2) a system-performance measures calculation using MIN and single-node queuing models.

4.1. Shared-Blocks Steady State Probabilities

The first step in our performance study is to calculate the shared-blocks steady state probabilities of the states imposed by the protocol. Transitions between the states are modeled as a discrete time Markov chain, as shown in Figure 3. For mathematical simplicity, we assume that all state transition times are equal. We introduce a *Not-Present* state for performance evaluation purposes. A block is in state Not-Present if it does not occupy an SCache frame. (A reference to a block in state Not-Present may require a write-back to

Table 1. Performance study parameters.

Input Parameters		
Parameter	Value	Units
f_r	0.75	—
f_w	0.25	—
h_{cp}	0.95	—
h_{cs}	variable	—
md	0.3	—
N_{sh}	variable	blocks
q_s	variable	—
S_b	variable	cycles
S_c	1	cycle
S_g	variable	cycles
S_l	variable	cycles
S_{mm}	10	cycles
S_{se}	2	cycles
z	2	cycles
ψ_{cp}	128	blocks
ψ_{cs}	256	blocks
Output Parameters		
U_b	Coherence bus utilization	
U_p	Processor utilization	

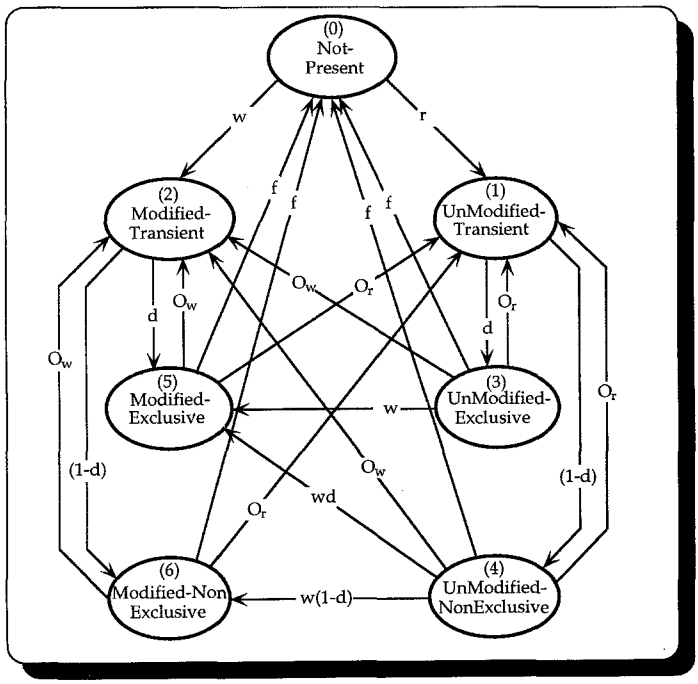

Figure 3. Shared-block states transition model.

to make room for the new block.) The seven states of the model are defined as Not-Present π_0, UnModified-Transient π_1, Modified-Transient π_2, UnModified-Exclusive π_3, UnModified-NonExclusive π_4, Modified-Exclusive π_5, and Modified-NonExclusive π_6. Let $P_{i,j}$ be the probability of transition from state i to state j in the discrete time Markov chain. The transition probabilities are given below.

$$P_{0,1} = r = U_p q_s f_r (zN_{sh})^{-1} \tag{2.a}$$

$$P_{1,3} = P_{2,5} = d = \pi_0^{N-1} \tag{2.b}$$

$$P_{1,4} = P_{2,6} = 1 - d = 1 - \pi_0^{N-1} \tag{2.c}$$

$$P_{0,2} = P_{3,5} = w = U_p q_s f_w (zN_{sh})^{-1} \tag{2.d}$$

$$P_{4,5} = wd = U_p q_s f_w \pi_0^{N-1} (zN_{sh})^{-1} \tag{2.e}$$

$$P_{4,6} = w(1 - d) = U_p q_s f_w (1 - \pi_0^{N-1})(zN_{sh})^{-1} \tag{2.f}$$

$$P_{3,0} = P_{4,0} = P_{5,0} = P_{6,0} = f = U_p q_s (1 - h_{cs})(z\psi_{cs})^{-1} \tag{2.g}$$

$$P_{3,2} = P_{4,2} = P_{5,2} = P_{6,2} = O_w = 1 - (1 - w)^{N-1} \tag{2.h}$$

$$P_{3,1} = P_{4,1} = P_{5,1} = P_{6,1} = O_r = (1 - w)^{N-1} - (1 - w - r)^{N-1} \tag{2.i}$$

In the above expressions, r(*read*) represents the probability of a shared-block read request; w(*write*) is the probability of a shared-block write request; f(*flush*) represents the probability of replacement of a shared block in an SCache; $d(= \pi_0^{N-1})$ is the probability that there are no copies of a block in remote caches; O_w is the probability of a remote write (a write request from any of the remaining $N - 1$ nodes); and O_r is the probability of a remote read request with no simultaneous remote write.

Equations for r and w are obtained as follows: A processor makes a request to one of its caches at the rate $U_p z^{-1}$, where U_p is the probability that the processor is busy and z is the average processor computation time. Since the probability of requesting a shared block (request directed to SCache) is q_s and the number of shared blocks in the system is N_{sh}, the probability that the request is to a certain shared block in the system is given by $q_s N_{sh}^{-1}$. Multiplying the term $(U_p q_s (z N_{sh})^{-1})$ with f_r or f_w gives the probability that a processor generates a shared-block read or write request, respectively. (Although an LRUSM reference model is used in our simulations, note that r and w contain a simple division by N_{sh}, which resembles an IRM model. This is because both reference models behave similarly in steady state despite having different block access probabilities in the near future [Trivedi 1982].)

The probability that an SCache block is chosen for replacement is given by ψ_{cs}^{-1}. As mentioned above, the term $U_p q_s z^{-1}$ gives the rate of a processor's request to its SCaches. The miss ratio of an SCache is $(1 - h_{cs})$. We derive O_w as follows: $(1 - W)^{N-1}$ is the probability that there is no write request from any of the remaining $N - 1$ nodes. Therefore, $1 - (1 - W)^{N-1}$ is the probability of at least one remote write request. Finally, O_r is obtained as follows: $1 - (1 - W - R)^{N-1}$ is the probability of at least one remote read or write request. The probability of at least one remote write request is given by $1 - (1 - W)^{N-1}$. Therefore, $[1 - (1 - W - R)^{N-1}] - [1 - (1 - W)^{N-1}] = [(1 - W)^{N-1} - (1 - W - R)^{N-1}]$ gives the probability of a read request with no simultaneous write request. Self loops in the Markov chain of Figure 3, which depict the situation when a block may stay in its current state, are not shown. The probability for each of these self loops can be obtained by subtracting all outgoing transition probabilities of a state from 1. (This satisfies the condition that the row sum of the transition matrix is 1 [Trivedi 1982].)

Equations 2.a to 2.i are solved for the steady state probabilities, π_i, of the Markov chain states, as given below.

$$\pi_0 = \left\{ 1 + \frac{O_r\gamma}{f} + \frac{O_w\delta}{f} + \frac{O_r\gamma}{f\theta} + \frac{1}{f(O_r + O_w + f)} \left[O_w\delta + \frac{O_r\gamma w}{\theta}(1 - d + 2d^2 - d^3) \right] \right\}^{-1} \tag{3.a}$$

$$\pi_1 = O_r\gamma\pi_0 f^{-1} \tag{3.b}$$

$$\pi_2 = O_w\delta\pi_0 f^{-1} \tag{3.c}$$

$$\pi_3 = O_r\gamma d\pi_0 (f\theta)^{-1} \tag{3.d}$$

$$\pi_4 = O_r\gamma(1 - d)\pi_0(f\theta)^{-1} \tag{3.e}$$

$$\pi_5 = [O_w\delta + O_r\gamma(2 - d)w\theta^{-1}][f(O_r + O_w + f)]^{-1}d\pi_0 \tag{3.f}$$

$$\pi_6 = [O_w\delta + O_r\gamma(1 - d)^2 w\theta^{-1}][f(O_r + O_w + f)]^{-1}(1 - d)\pi_0 \tag{3.g}$$

where $\gamma = r + w - rf\alpha^{-1}$, $\delta = r + w - wf\alpha^{-1}$, and $\theta = O_r + O_w + f + w$.

4.2. MIN and Single-Node Queuing Models

In order to compute system-performance measures in terms of processing power and bus utilization, a hierarchical (two-level) performance evaluation study is carried out. A queuing model of a complete MIN-based system incorporating N nodes and forward and backward MINs is given in [Bhuyan et al. 1989]. Here, we take a simpler approach. First, we model only the MIN (in this paper, the Butterfly) as a queuing network and solve it to find the average delay to access data from a remote memory module S_g and a local memory module S_l, with the request rate as the input parameter. Then we develop a queuing model for a single node using these average delays. The model depicts the behavior of a processor's request (whether it is private or shared, hit or miss, local or remote, etc.) in the system with cache coherence enforced. The single-node model is based on the assumption that all system nodes perform identical tasks. The relaxation of this assumption can be captured by the model, first, by using the appropriate input rates to the MIN model for finding new S_g and S_l, and then by evaluating the node model separately for different types of computation.

A representative queuing model for a Butterfly MIN, made up of 4 × 4 switching elements (SE) with 16 nodes, is shown in Figure 4. Each switch is represented as four queues. The network is modeled as a forward and backward MIN to represent the two-pass communication protocol used in the BBN Butterfly. For mathematical simplicity we assume that the SEs have an infinite buffering capability. We solve the MIN model using the open network queuing technique [Lazowska et al. 1984]. The input parameters to the model are network size N, SE service time, and memory access time. The model can handle uniform or favorite memory requests (with higher probability for a certain memory module than others). Here, we are using a uniform reference model for simplicity of description. A detailed description of the solution is not included here since it is straightforward and similar to the approach used in [Bhuyan et al. 1989]. We compute the average delays S_g and S_l from this model. Figure 5 shows the variation of S_g and S_l with a traffic rate for a Butterfly-type network of size 16 × 16. We also compute S_g and S_l for different system sizes. The novelty of this approach is that for any traffic generated by a node, average delays, S_g and S_l, are already known from the MIN model. Further, the MIN model is solved only once.

A queuing model of a node is next developed using these delays, as shown in Figure 6. A processor does internal computation for a certain amount of time z, followed by a request to one of its caches with a rate λ_p. Requests are directed to the PCache with a rate $\lambda_{pp}(= q_p\lambda_p)$ and to the SCache with a rate $\lambda_{ps}(= q_s\lambda_p)$. Traffic due to a PCache hit is given by the rate λ_{chp} and by λ_{chs} for an SCache hit. The processor then resumes

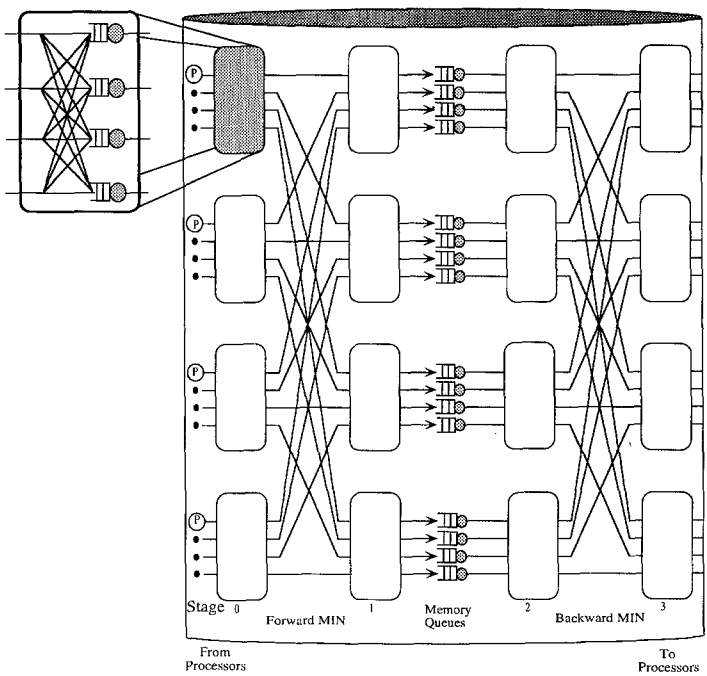

Figure 4. A queuing model for a 16-node Butterfly MIN with 4 × 4 switching elements.

computation after a delay related to the service time of the relevant cache S_c. (It is assumed that both caches have similar service times.) Traffic due to a PCache miss is with a rate λ_{cp}; a miss is satisfied by a block fetch either from the local or a remote memory module through the MIN. Due to our uniform referencing assumption, PCache miss requests go to the local memory of a node at a rate $\lambda_{cpl} = \lambda_{cp}N^{-1}$ and to other memory modules at a rate $\lambda_{cpg} = (N - 1)N^{-1}\lambda_{cp}$. Note that this single-node queuing model can be easily adapted to take care of "favorite" memory references, where requests sent to memory modules are controlled by some probability. Requests with higher probability are directed to the local memory module of the node.

The coherence control bus is represented as a queuing center with service time S_b. Requests serviced by this queuing center include coherence signals and write-update broadcasts, given by the rate λ_{cb}. After doing write-update broadcasts, which is given by the rate λ_{chb}, the processor resumes computation. If a request is a shared data miss, a signal is broadcast over the coherence bus for the state of the block. The block would be fetched from memory, either directly (memory still has the valid copy) with a rate λ_{cm}, or after writing back the updated copy of the block from a remote cache given by the rate λ_{cr}. The distribution of requests to the local and remote memory modules is similar to that for private blocks. The write-back of the shared block to a memory module and the acknowledgment incur a delay of S_g (if the write-back is remote) or S_l (if the write-back is local). The rates for both cases are given as λ_{crg} and λ_{crl}, respectively. The cache controller that performed the write-back broadcasts the acknowledgment on the bus. Note

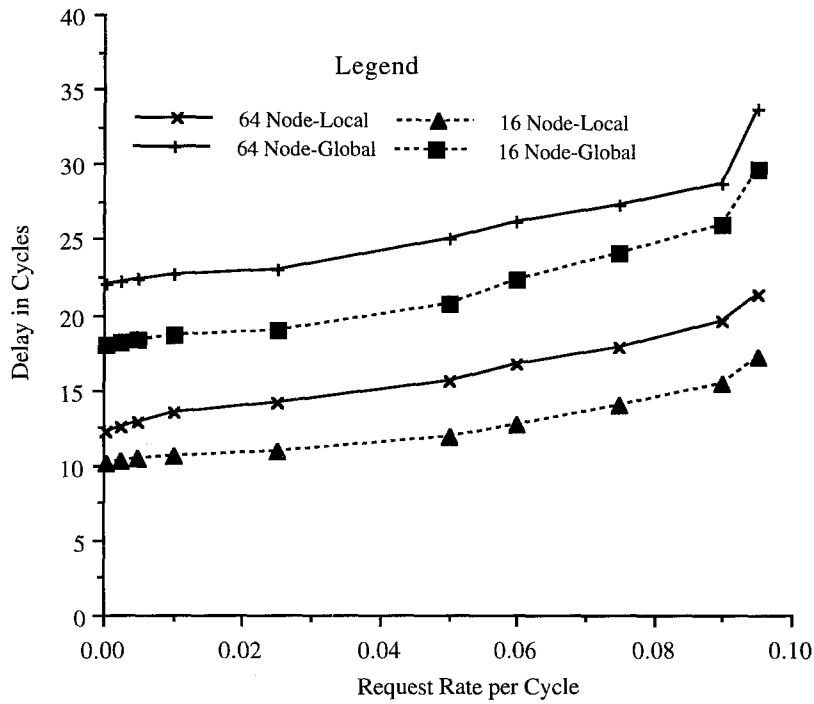

Figure 5. S_l and S_g delays versus input traffic for 16- and 64-node Butterfly MINs.

that after a remote write-back, the block fetch could be either local or remote. Let these two rates be represented as λ_{crgl} and λ_{crgg}, respectively. The block fetch experiences a global delay, S_g, when the write-back is local. Note that the same bus queue has been shown at several places in Figure 6 in order to depict bus accesses clearly at different phases of the protocol.

Write-back and block-fetch requests are represented as open-class customers. The contribution of the open-class customers is to inflate the service times of various centers since the model is mixed. Write-back operations increase the traffic in the MIN by the rate λ_{wb} — this includes the effect on the local and remote memory modules, which are represented as λ_{wbl} and λ_{wbg}, respectively. Traffic on the bus from the other $N - 1$ nodes is represented by λ_b. The effect of loading a block and writing back blocks from the PCache and SCache are represented as open-class customers with rates λ_{lp} and λ_{ls}, respectively.

The above model is solved using the mean value analysis (MVA) technique to calculate the bus and processor utilizations [Lazowska et al. 1984]. The processor utilization, U_p, depends on shared-blocks steady state probabilities. These steady state probabilities in turn need U_p as a parameter. Because of this interdependence, we solve for shared blocks steady state probabilities starting with $U_p = 1$. These probabilities are in turn used to solve the node model to obtain the processor and bus utilizations. The calculated processor utilization is used again to compute the steady state probabilities, which are later used to solve the model again. This is repeated until U_p converges. Coherence bus utilization, U_b, is

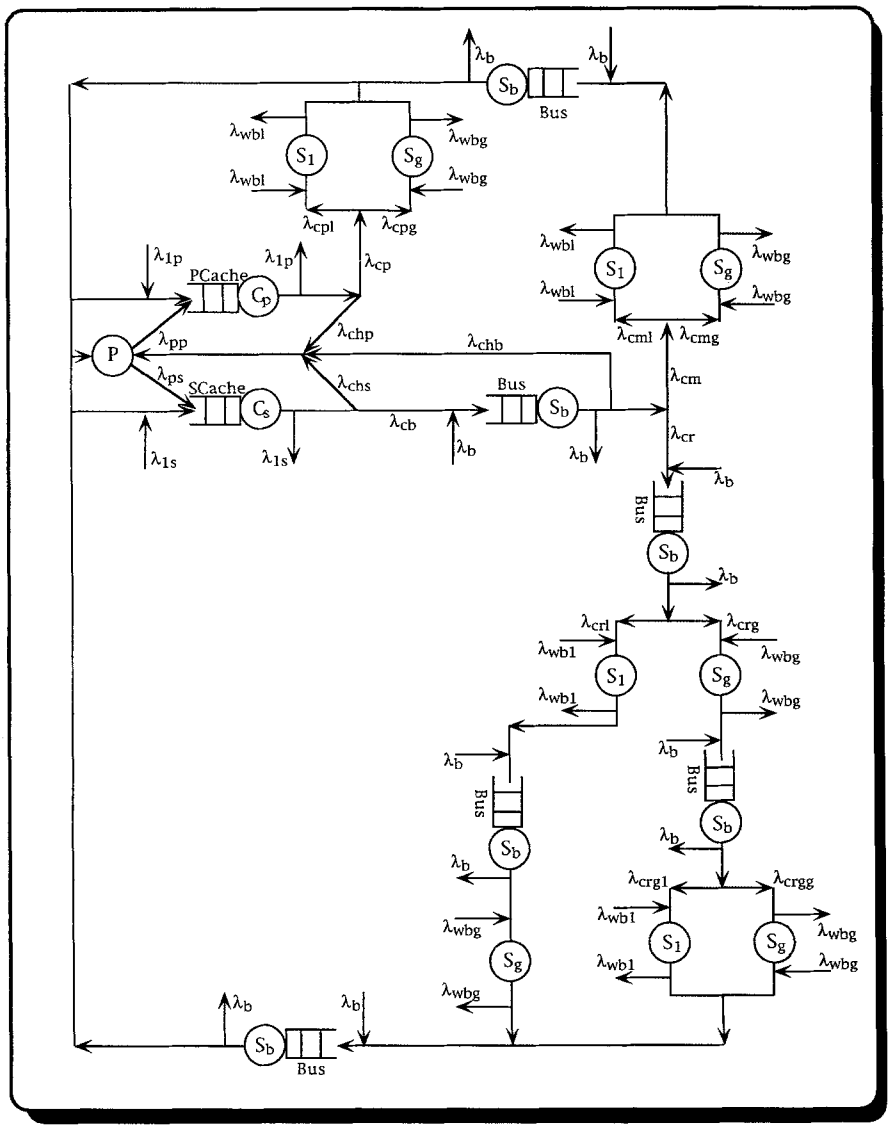

Figure 6. A queuing model for a single node in the system.

obtained from the node model by considering the effect of both open- and closed-class customers on the bus. The coherence bus utilization due to open-class customers, U_{bo}, is found from the bus traffic λ_b as $U_{bo} = \lambda_b S_b$. Note that λ_b represents bus traffic from $N - 1$ nodes. Due to the homogeneity of all nodes, total coherence bus utilization is approximated by $U_b = N(N - 1)^{-1} U_{bo}$.

The single-node queuing model is solved for a BBN Butterfly-like multiprocessor system, with a MIN made up of 4 × 4 switching elements. Table 1 shows the input parameters

for this performance study and the output performance measures. Notice that the PCache hit ratio, h_{cp}, is constant, 0.95. However, the SCache hit ratio, h_{cs}, is variable; it is computed from the steady state probabilities as $h_{cs} = 1 - \pi_0 - \pi_1 - \pi_2$ (a request is an SCache miss if the block is in state Not-Present, Modified-Transient, or UnModified-Transient). It is assumed that the service time of both caches, $S_c = 1$ *cycle*. The bus service time, S_b, is assumed to be 1, 2, and 4 cycles for systems with 16, 64, and 256 nodes, respectively. Since it is common that larger systems deal with larger amounts of shared data blocks, we assume the number of shared blocks in the system to be 128, 512, and 2048 for systems of sizes 16, 64, and 256, respectively. PCache and SCache sizes should be a function of system size; however, for simplicity we assume that they remain the same for the range of system nodes selected for this performance study.

Figures 7 and 8 show the coherence bus and processor utilization variation with the degree of sharing for three system sizes. The degree of sharing, q_s, is the probability that a processor makes a request to a shared block in the system. For practical systems, this probability is small and ranges from 5% to less than 20% of the total processor requests. Note that our node model can handle any degree of sharing. Therefore, theoretically, we should be able to provide performance measures for any degree of sharing between 0% and 100%. We refrained from doing so in order to keep our results as practical and realistic as possible.

Bus utilization, U_b, increases as the degree of sharing increases due to more coherence control transactions on the bus. (Note that for a degree of sharing of 0%, U_b will be zero

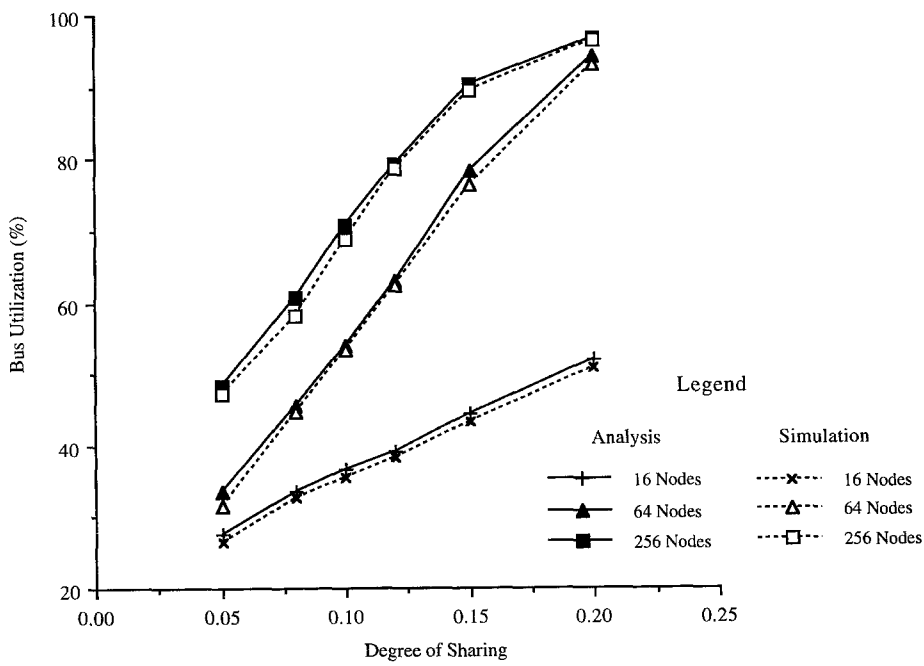

Figure 7. Coherence bus utilization (%) versus degree of sharing for selected system sizes.

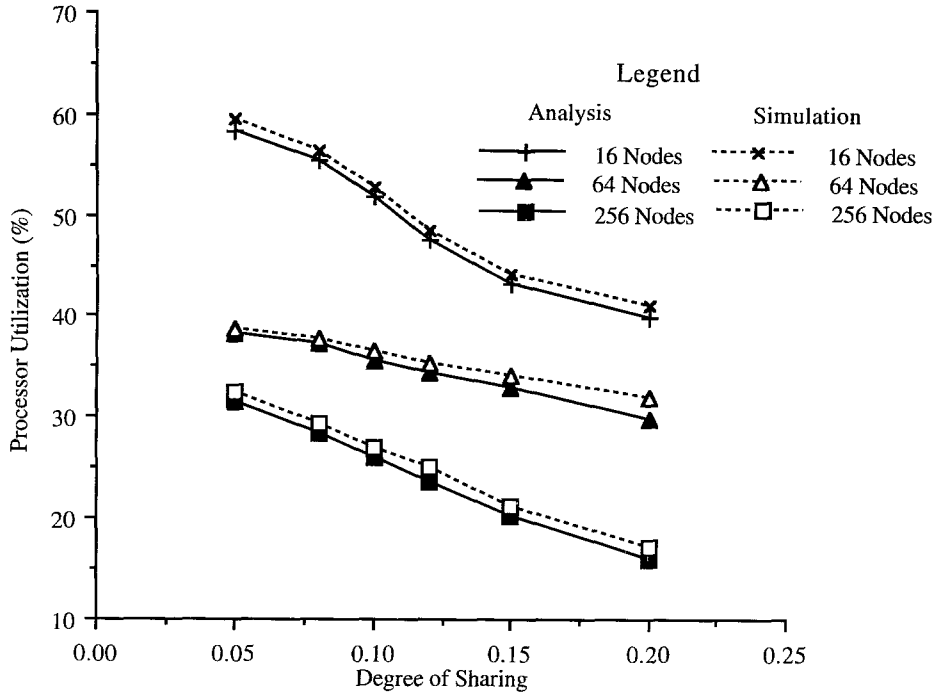

Figure 8. Processor utilization (%) versus degree of sharing for selected system sizes.

since there will be no requests to shared blocks and hence neither a data inconsistency problem nor any traffic on the bus.) A degree of sharing of 100% is completely unrealistic since it confines all processor requests to shared blocks only. The coherence bus will saturate for much smaller degrees of sharing, as shown in Figure 7. The processor utilization, U_p, decreases as the degree of sharing increases due to more coherence control transactions and therefore more waiting time for the processor.

It is clear from Figure 7 that our scheme can be implemented on medium-sized multiprocessors. For a system with 256 nodes, bus utilization is practical for degrees of sharing of about and less than 10%, which is the common practical range for the degree of sharing. Higher degrees of sharing tend to saturate the bus. The reason our scheme scales to systems having up to 256 nodes is the minimal amount of traffic on the bus. We only allowed coherence control signals on the bus; no memory block transfer is allowed on the bus. This boosted the scalability to 256 nodes.

In order to provide a comparison between this architecture and an equivalent unified-cache architecture, we simulated the coherence protocol on a unified-cache architecture. It is assumed that the unified-cache size is equivalent to the sum of both the PCache and SCache in the split-cache system. We found that the coherence bus utilization is reduced in the split-cache architecture by around 10% for a system with 256 nodes, as shown in Figure 9. This is mainly due to the interference between shared and private blocks at cache block replacement in a unified cache, as explained earlier. Figure 10 compares the processor utilization between a split-cache and a unified-cache system. It is observed that the processor utilization increases in the split-cache system by about 8%. This is attributed to less traffic on the bus as well as fewer replacements due to the splitting of the cache.

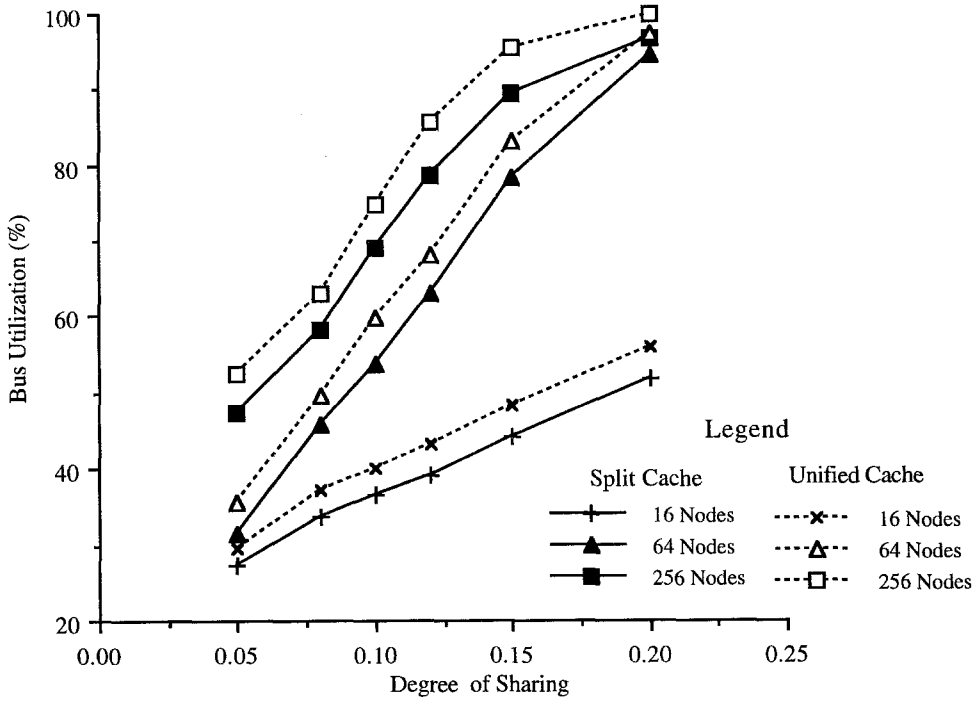

Figure 9. Coherence bus utilization (%) versus degree of sharing for split-cache and unified-cache architectures.

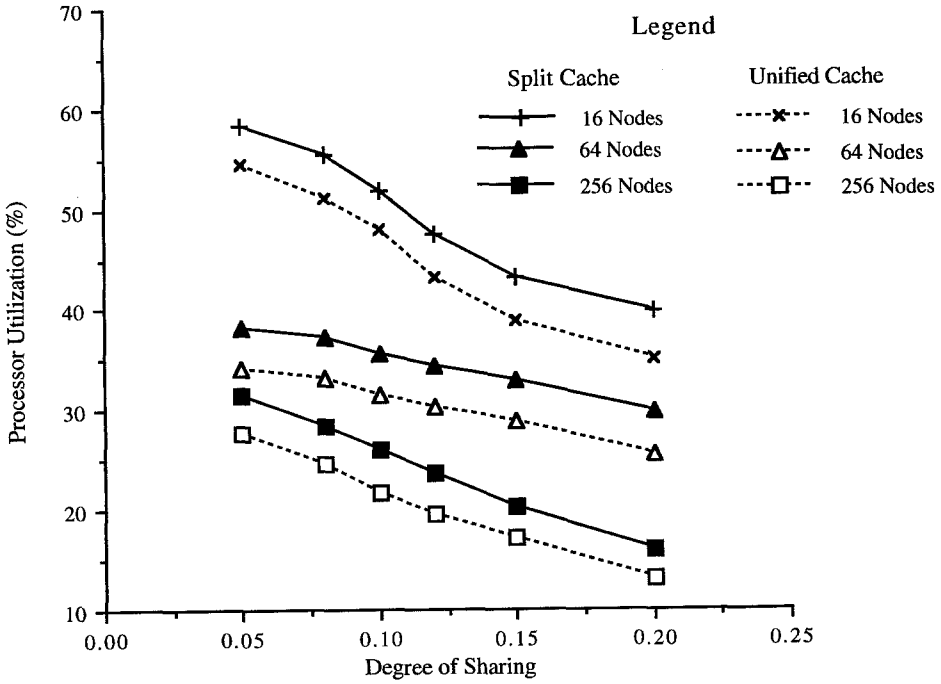

Figure 10. Processor utilization (%) versus degree of sharing for split-cache and unified-cache architectures.

In order to provide a feel for how well our system performs, we compared the processor utilization of our architecture with a similar architecture (without the coherence control bus) that implements a full map directory scheme [Censier and Feautrier 1978]. (We did not compare the performance of our architecture with an actual commercial machine due to the lack of such large-sized machines, up to 256 nodes, with a cache coherence scheme.) The full map directory scheme includes a pointer to each cache that has a copy of a block and also includes one Modified/UnModified bit. Updates on a block are sent point to point through the MIN to update copies of the block in other caches. We simulated this architecture with the cache coherence enforced and measured the processor utilization, as shown in Figure 11. It is clear that the contribution of the bus/split cache is significant—an increase in processor utilization up to 20% is observed.

5. Conclusions

In this paper a write-update cache coherence protocol for shared-memory split-cache MIN-based multiprocessors has been proposed and evaluated. Each node has two dedicated private caches: a PCache for blocks private to a process and an SCache for blocks shared among all processes. To implement the protocol, we extended the system architecture by adding a coherence control bus connecting all SCache controllers. Four specific design issues were addressed: (1) proposing a suitable bus-based protocol that puts a minimal burden on the

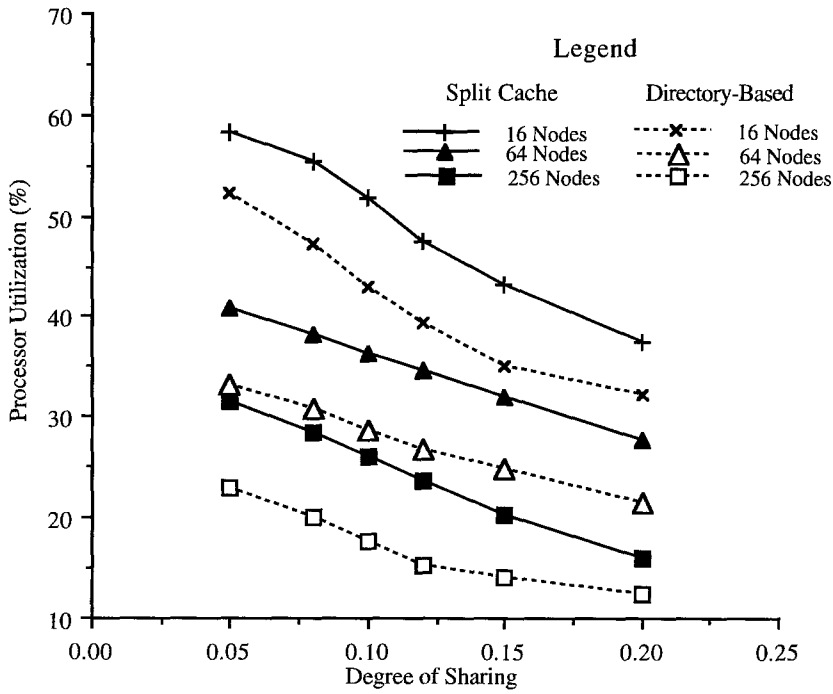

Figure 11. Processor utilization (%) comparison versus degree of sharing.

bus for improving system scalability, (2) avoiding timing problems due to the variable MIN delay, (3) analyzing the coherence protocol quantitatively to compute system performance, and (4) determining the parameter value ranges under which this approach is feasible.

Scalability is a key concern in multiprocessor design. It has been shown in this work that a medium-sized MIN-based multiprocessor with up to 256 nodes can be implemented using our approach to resolve cache coherence. Limiting the amount of traffic on the coherence bus is the main factor behind the ability to approach such a system size. This split-cache approach has been evaluated extensively by comparing it to an equivalent-sized unified cache and to a multiprocessor with a directory-based cache coherence scheme.

Appendix: Single-Node Queuing Model Equations

A processor makes a request to a cache with a rate equivalent to $U_p z^{-1}$. The private request rate (request to the PCache) is given as $\lambda_{pp}(= q_p U_p z^{-1})$, and the shared request rate is $\lambda_{ps}(= q_s U_p z^{-1})$. A request flows in the model of Figure 6 with different rates, as shown below.

λ_{chp} (traffic rate due to a PCache hit):

$$\lambda_{chp} = h_{cp}\lambda_{pp}.$$

λ_{chs} (traffic rate due to an SCache hit):

$$\lambda_{chs} = h_{cs}\lambda_{ps}.$$

λ_{cp} (traffic rate due to a PCache miss):

$$\lambda_{cp} = (1 - h_{cp})\lambda_{pp}.$$

λ_{cb} (traffic rate for shared requests that require using the bus):

$$\lambda_{cb} = [(1 - h_{cs}) + f_w(\pi_4 + \pi_6)]\lambda_{ps}.$$

λ_{chb} (traffic rate for shared requests that require only a broadcast over the bus):

$$\lambda_{chb} = f_w(\pi_4 + \pi_6)\lambda_{ps}.$$

λ_{cpl} (traffic rate due to a PCache miss served by the local memory module):

$$\lambda_{cpl} = N^{-1}\lambda_{cp}.$$

λ_{cpg} (traffic rate due to a PCache miss served by a remote memory module):

$$\lambda_{cpg} = (N - 1)N^{-1}\lambda_{cp}.$$

λ_{cr} (traffic rate due to an SCache miss served by an updated copy in a remote SCache):

$$\lambda_{cr} = (1 - h_{cs})[1 - (1 - \pi_5 - \pi_6)^{N-1}]\lambda_{ps}.$$

λ_{cm} (traffic rate due to an SCache miss served by a memory module directly):

$$\lambda_{cm} = (1 - h_{cs})[(1 - \pi_5 - \pi_6)^{N-1}]\lambda_{ps}.$$

λ_{crl} (traffic rate due to an SCache miss served by the local memory module after writing back the block from a remote SCache):

$$\lambda_{crl} = N^{-1}\lambda_{cr}.$$

λ_{crg} (traffic rate due to an SCache miss served by a remote memory module after writing back the block from a remote SCache):

$$\lambda_{crg} = (N - 1)N^{-1}\lambda_{cr}.$$

λ_{crgl} (traffic rate due to an SCache miss served by a local memory module after writing back the updated block from an SCache to a remote memory module):

$$\lambda_{crgl} = N^{-1}\lambda_{crg}.$$

λ_{crgg} (traffic rate due to an SCache miss served by a remote memory module after writing back the updated block from a remote SCache to a remote memory module):

$$\lambda_{crgg} = (N - 2)N^{-1}\lambda_{crg}.$$

λ_{cml} (traffic rate due to an SCache miss served by a local memory module):

$$\lambda_{cml} = N^{-1}\lambda_{cm}.$$

λ_{cmg} (traffic rate due to an SCache miss served by a remote memory module):

$$\lambda_{cmg} = (N - 1)N^{-1}\lambda_{cm}.$$

As mentioned before, the network is a mixed model with open- and closed-class customers. Open-class customers are generated due to the following traffic.

λ_{wb} (traffic rate due to a write-back operation, which increases the traffic in the MIN; this traffic is generated due to a PCache or SCache miss, and the selected block is modified, as given in the following equation):

$$\lambda_{wb} = (N - 1)[(1 - h_{cp})q_p md + (1 - h_{cs})q_s(\pi_5 + \pi_6)]\lambda_p.$$

λ_{wbl} (traffic rate due to a write-back directed to the local memory module):

$$\lambda_{wbl} = N^{-1}\lambda_{wb}.$$

λ_{wbg} (traffic rate due to a write-back directed to a remote memory module; this increases the delay in the MIN):

$$\lambda_{wbg} = (N - 1)N^{-1}\lambda_{wb}.$$

λ_b (traffic rate due to write broadcasts $q_s f_w(\pi_4 + \pi_6)\lambda_p$; broadcasts to update local directories due to a miss or write-back; and broadcasts due to a block being in a Transient state):

$$\lambda_b = (N - 1)\{f_w(\pi_4 + \pi_6) + (1 - h_{cs})[1 + \pi_1 + \pi_2]\}\lambda_{ps}.$$

λ_{lp} (traffic due to loading a private block in the PCache due to a miss):

$$\lambda_{lp} = (1 - h_{cp})\lambda_{pp}.$$

λ_{ls} (traffic due to loading a shared block in the SCache due to a miss and write-backs of modified blocks):

$$\lambda_{ls} = \left\{(1 - h_{cs})\left[1 + \frac{f_w}{N_{sh}}(\pi_5 + \pi_6)\right] + \frac{(\pi_5 + \pi_6)}{N_{sh}}\right\}\lambda_{ps}.$$

The effect of open-class customers is to inflate the service demand of the centers they run through and hence degrade system performance. Using the equations mentioned above, we compute the request response time using the standard mixed model MVA technique [Lazowska et al. 1984]. However, the solution needs the branching probabilities to all centers, which are obtained by substituting $\lambda_p = 1$. Note that this mixed model has only one closed-class customer, which is a processor's request.

References

Agarwal, A., Lim, B.-H., Kranz, D., and Kubiatowicz, J. 1990. APRIL: A processor architecture for multiprocessing. In *Conf. Proc.—The 17th Annual Internat. Symp. on Comp. Architecture* (Seattle, May 28–31), pp. 104–114.

Archibald, J.K. 1987. The cache coherency problem in shared memory multiprocessors. Ph.D. thesis and tech. rept. 87-02-06, Dept. of Comp. Sci., Univ. of Wash., Seattle.

BBN. 1989. *Butterfly GP1000 Switch Tutorial.* BBN Advanced Computers, Inc.

Bhuyan, L.N., Liu, B., and Ahmed, I. 1989. Analysis of MIN based-multiprocessors with private cache memories. In *Conf. Proc.—The 1989 Internat. Conf. on Parallel Processing* (St. Charles, Ill., Aug. 14–18), Penn. State Univ. Press, vol. 1, pp. 51–58.

Censier, L.M., and Feautrier, P. 1978. A new solution to coherence problems in multicache systems. *IEEE Trans. Comps.*, C-27, 12 (Dec.): 1112–1118.

Cheong, H., and Veidenbaum, A.V. 1988. A cache coherence scheme with fast selective invalidation. In *Conf. Proc.—The 15th Annual Internat. Symp. on Comp. Architecture* (Honolulu, May 30–June 2), pp. 299–307.

Dubois, M., and Briggs, F.A. 1982. Effects of cache coherency in multiprocessors. *IEEE Trans. Comps.*, C-31, 11 (Nov.): 1083–1099.

Eggers, S. 1989. Simulation analysis of data sharing in shared memory multiprocessors. Ph.D. thesis and tech. rept. UCB/CSD 89/501, Univ. of Calif., Berkeley.

Goodman, J.R., and Woest, P.J. 1988. The Wisconsin multicube: A new large-scale cache-coherent multiprocessor. In *Conf. Proc.—The 15th Annual Internat. Symp. on Comp. Architecture* (Honolulu, May 30–June 2), pp. 422–431.

Gustavson, D.B. 1992. The scalable coherent interface and related standard projects. *IEEE Micro*, 12, 2 (Feb.): 10–22.

Lazowska, E., Zahorjan, J., Graham, G.S., and Sevcik, K.C. 1984. *Quantitative System Performance*. Prentice-Hall, New York.

Lenoski, D., Laudon, J., Gharachorloo, K., Gupta, A., and Hennessy, J. 1990. The directory-based cache coherence protocol for the DASH multiprocessor. In *Conf. Proc.—The 17th Annual Internat. Symp. on Comp. Architecture* (Seattle, May 28–31), pp. 148–159.

Min, S.L., and Baer, J.L. 1989. A timestamp-based cache coherence scheme. In *Conf. Proc.—The 1989 Internat. Conf. on Parallel Processing* (St. Charles Ill., Aug. 14–18), Penn. State Univ. Press, vol. 1, pp. 23–32.

Mizrahi, H.E., Baer, J.L., Lazowska, E.D., and Zahorjan, J. 1989. Extending the memory hierarchy into multiprocessor interconnection networks: A performance analysis. In *Conf. Proc.—The 1989 Internat. Conf. on Parallel Processing* (St. Charles, Ill., Aug. 14–18), Penn. State Univ. Press, vol. 1, pp. 41–50.

Smith, A.J. 1982. Cache memories. *ACM Comp. Surveys*, 14, 3 (Sept.): 473–530.

Smith, A.J. 1985. CPU cache consistency with software support and using one time identifiers. In *Conf. Proc. of the 1985 Pacific Comp. Commun. Symp.* (Seoul, S. Korea, Oct. 22–24), pp. 153–161.

Stenström, P. 1989. A cache consistency protocol for multiprocessors with multistage networks. In *Conf. Proc.—The 16th Annual Internat. Symp. on Comp. Architecture* (Jerusalem, Israel, May 28–June 1), pp. 407–415.

Tang, C.K. 1976. Cache system design in the tightly coupled multiprocessor system. In *Conf. Proc.—The 1976 AFIPS Nat. Comp. Conf.*, pp. 749–753.

Trivedi, K.S. 1982. *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, New York.

Yen, W.C., and Fu, K.S. 1982. Analysis of multiprocessor cache organizations with alternative main memory update policies. In *Conf. Proc. of the 9th Annual Internat. Symp. on Comp. Architecture* (Austin, Tex., Apr. 26–29), pp. 89–100.

Yousif, M.S. 1991. Effective use of caches in MIN-based multiprocessors. Ph.D. thesis, Dept. of Electrical and Comp. Engineering, Penn. State Univ., University Park, Penn.