Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets

Mary K. Vernon and Mark A. Holliday
Computer Sciences Department
University of Wisconsin — Madison
Madison, WI 53706

Abstract

We use an exact analytical technique, based on Generalized Timed Petri Nets (GTPNs), to study the performance of *shared bus cache consistency protocols* for multiprocessors. We develop a general framework within which the key characteristics of the Write-Once protocol and four enhancements that have been combined in various ways in the literature can be identified and evaluated. We then quantitatively assess the performance gains for each of the four enhancements. We consider three levels of data sharing in our workload models. One of the enhancements substantially improves system performance in all cases. Two enhancements are shown to have neglibile effect over the range of workloads analyzed. The fourth enhancement shows a small improvement for low levels of sharing, but shows more substantial improvement as sharing is increased, if we assume a "good access pattern". The effects of two architectural parameters, the *blocksize* and the *main memory cycle time* are also considered.

Section 1. Introduction

Simulation is the traditional technique for performance evaluation when a detailed characterization of the system is needed. We, however, have recently developed a technique [HOL85a, HOL85b] which can be used to obtain exact analytical results far more quickly, if the state space is not too large. In our technique, we model the system using a version of Petri Nets that we call Generalized Timed Petri Nets (GTPNs). The GTPN differs from the Stochastic Petri Net (GSPN) models [NAT80, MOL82, AJM84, MOL85] in that transition firing durations are deterministic rather than stochastic. However, the GTPN can be viewed as a stochastic process because a probability distribution is defined over the possible next states. We find the state space of the Markov Chain embedded within a GTPN model and derive performance measures, automatically. We have shown that deterministic firing durations are useful for modeling memory access times in multiprocessors [HOL85b]. Geometric holding times can also be represented in the GTPN, for example to model memory interrequest times. The comparison of cache consistency protocols in this paper further illustrates the utility of our technique.

MIMD multiprocessors with multiple independent memory modules and a single stage multibus interconnection network form an important class of architectures. To minimize contention for the shared global memory modules and the buses, the processors are usually assumed to also have local memories. If the local memories are used as caches, the problem of maintaining consistency among the caches immediately arises [SMI82]. Two classes of dynamic cache consistency protocols have developed. The first class allows a general interconnection network but requires that a global directory be maintained by the shared memory modules. The second class requires that the interconnection network be a shared bus, but cache consistency is maintained in a distributed manner by the caches. We are interested in comparing the performance of protocols in this second class: the *shared bus cache consistency protocols.*

Several shared bus cache consistency protocols have recently been proposed [GOO83, MCC84, PAP84, RUD84, KAT85]. The Write-Once protocol, designed for the Multibus, was the first protocol to appear. Other proposals since then have suggested modifications to the Write-Once protocol which may improve system performance. Several of the proposals contain some analysis of the expected performance gains for the proposed protocol. However, it is not clear which of the modifications within a protocol are primarily responsible for the performance improvement. In some cases, the proposed modification requires a more complex (i.e. more expensive) bus or cache controller. A study is needed to determine how much increase in performance can be expected for each proposed modification. Archibald and Baer's simulation study [ARC85] is the most comprehensive performance comparison of the published protocols to date. In particular, they provide a uniform description of the protocols and they identify the important differences between the protocols. We propose in this paper a more precise model of the protocols which improves the relevance of the performance comparison. Furthermore, instead of modeling the protocols proposed in the literature exactly, we isolate four key enhancements to Write-Once which have been combined in various ways in these protocols. We then study the performance gains for each of the four enhancements.

In section 2 we describe our assumptions about the general characteristics of the multiprocessor. In section 3 we describe a Basic protocol similar to Write-Once, four enhancements to the Basic protocol, and how these enhancements are combined in protocols proposed in the literature. In section 4 we describe our GTPN models of the Basic and enhanced protocols. In section 5 we present our results, and in section 6 we summarize our work.

Section 2. Multiprocessor Characteristics

Figure 2.1 illustrates the multiprocessor configuration. Each processor has one local cache. A single shared bus connects the caches and the main memory. A processor is directly connected only with its local cache. Each local cache is directly connected to its own processor and, through the shared bus, with main memory and all the other caches.

The cycle times of the processors, caches, and the bus are the same and define the basic time unit. The main memory cycle time is four of these basic time units. Below the term *cycle* refers to the basic time unit.

This research was partially supported by the National Science Foundation under grants DCR-8402680 and DCR-8451405, and by grants from IBM and Cray Research, Inc.

© 1986 ACM 0-89791-184-9/86/0500-0009 $00.75

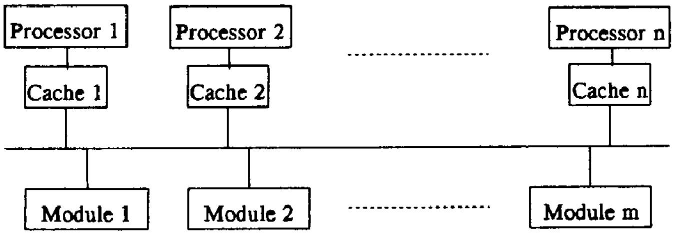

Figure 2.1. The Multiprocessor Configuration

Section 2.1. Operation of the Memory and Bus

The cache memories and main memory are divided into *words* which are organized into *blocks*. A cache entry is a physical location within cache memory that consists of: 1) the cache's copy of a memory block, and 2) a state. The number of words in a block is the *blocksize*. The main memory is divided into *blocksize* modules interleaved on the low-order address bits.

A block is transferred to or from main memory by a sequence of transfers of one word per bus cycle. The words in a block are always transferred in the same order, as opposed to transferring the needed word first for a memory read operation. On a main memory write, the bus is released as soon as the word(s) are transferred to memory, although the memory module(s) will be busy for additional cycles to complete the write operation. Thus, memory contention can occur although there is only one bus.

An arbitrary number of the caches can simultaneously read the information on the bus.

Section 2.2. Operation of the Cache

The cache is used for both instructions and data. When a processor makes a cache request, it holds until the request is satisfied. A cache request is either a read (CPU-READ) or write (CPU-WRITE) for a particular word that is in a block that might or might not have a valid copy in the local cache. If the processor's cache can service the request locally and immediately, then servicing takes one cycle.

A cache has two independent parts: a bus monitor and a controller. For every bus transaction, each cache's bus monitor determines if the referenced block has a valid copy in the local cache. In the case of such a block match, the cache's bus monitor signals the cache controller.

In a given cycle a cache controller can simultaneously receive a request from its processor and a signal from its bus monitor requiring action. To avoid a race condition, the controller can service only one of the requests at a time. The bus monitor request always has priority. Similarly, loading a block from the bus cannot be done simultaneously with the cpu read or write operation that initiated a load request.

There are four types of bus transactions: READ, READ-MOD, WRITE, and INVALIDATE. A READ (READ-MOD) transaction is started by a cache that has a miss on a read (write) from its processor. A WRITE transaction is due to a cache writing a word to main memory via the bus. An INVALIDATE transaction is started by a cache that wants to invalidate all the copies other caches have of a particular block. For all four types of transactions, the main memory address of the affected block is on the bus's address lines.

Each bus transaction type has an associated bus control line. There is one other bus control line, *shared*. It can be raised by more than one cache simultaneously. A cache raises the *shared* line to announce that it has a valid copy of the block involved in the transaction.

The READ and READ-MOD transactions implicitly involve main memory supplying a block. However, it is possible for any cache to inhibit main memory and to respond instead. Arbitration schemes determine which cache will respond if more than one can, and which cache gains control of the bus to initiate a new transaction.

Section 2.3. States of the Cache Entries

Each cache entry has three bits of state information. The first bit indicates if the entry contains a valid or invalid copy of a main memory block. The second and third bits implement the two binary attributes that define the state of a valid copy of a block: *number of copies*, and *need to write back*. *Number of copies* can be ONLY or NOT-ONLY. ONLY means that the cache knows that it is the only cache with a valid copy. NOT-ONLY means that the cache does not know that ONLY holds. Other valid copies might or might not exist. *Need to write back* is WBACK or NO-WBACK. WBACK means that on replacement this cache has to write-back to main memory its copy of the block. The WBACK bit is a generalization of the traditional "dirty" bit, which indicates that the contents of the block have been written since the last time the block was written to main memory. In some of the more sophisticated protocols, multiple valid copies may exist of a "dirty" block, yet only one of these copies in state WBACK.

Section 3. The Protocols

We define a Basic protocol which does not use the *invalidate* or *shared* control lines. We then describe four enhancements to the Basic protocol. In Section 3.3, we describe how the various protocols proposed in the literature are related to the Basic protocol and the four enhancements.

The protocols are specified by defining the actions taken by a cache in response to requests from the bus or local processor. Actions are defined for the type of request, and the state of the relevant block copy in the local cache memory. Recall that bus transactions have priority over processor requests. Thus a processor request is blocked when there is a bus transaction that matches a valid local block copy. Lost cycles due to cache, bus, or memory contention are not mentioned in the protocol definition.

Section 3.1. The Basic Protocol

The cache takes the actions 1 and 2 in response to bus transactions:

1) READ or READ-MOD

   a) No valid copy: Do nothing. If no cache has a valid copy, then main memory supplies the block in 3 + *blocksize* cycles.

   b) (ONLY or NOT-ONLY, NO-WBACK): One of the caches is chosen to supply the block and main memory is inhibited. (Note that no other cache can have a copy in state (ONLY, WBACK)). On the cycle after the bus request, the supplier puts the block on the bus for *blocksize* cycles.

      The block copies in all these caches go to state (NOT-ONLY, NO-WBACK) or are invalidated for a READ and READ-MOD, respectively.

   c) (ONLY, WBACK): At most one cache can be in this block substate. It inhibits main memory. On the cycle after the bus request, the supplier puts the block on the bus for *blocksize* cycles. It then writes the block to main memory for *blocksize* cycles. The block copy goes to state (NOT-ONLY, NO-WBACK) or is invalidated for a READ and READ-MOD, respectively. Note that (NOT-ONLY, WBACK) is impossible in the Basic protocol.

2) WRITE

a) Any valid copy: Block copy is invalidated.

Actions 3 and 4 are taken in response to processor requests:

3) CPU-READ

a) Any valid copy ("Read Hit"): Spend 1 cycle locally supplying the processor.

b) No valid copy ("Read Miss"): In the current cycle put a READ on the bus with the desired block address. For the *blocksize* (+3) next cycles the supplier (another cache or main memory) supplies the block. When the block has been supplied, then if the supplier was a cache block in state (ONLY, WBACK), then the supplier will use the bus for the next *blocksize* cycles for write-back to main memory. When the supplier is finished with the bus, if a requester's cache entry in state (ONLY, WBACK) had to be replaced, then write it to main memory in *blocksize* cycles. When the requester is finished with the bus, then in the next cycle it supplies the desired word to its processor. Set block copy state to (NOT-ONLY, NO-WBACK).

4) CPU-WRITE

a) No valid copy: Same as 1.a. except for three changes: 1) at the start READ-MOD is raised, 2) the last cycle of servicing the request is an update in the cache instead of a processor read, and 3) the new block state is (ONLY, WBACK).

b) (ONLY, NO-WBACK or WBACK) Spend 1 cycle locally updating the word. Set block copy to state (ONLY, WBACK).

c) (NOT-ONLY, NO-WBACK) Spend 1 cycle updating the word locally. Then spend 1 cycle using the bus to write the word to main memory with the WRITE line raised. This will be referred to as the *broadcast write*. Set block copy state to (ONLY, NO-WBACK).

Section 3.2. Enhancements

We consider four independent enhancements. The first three reduce the times blocks or words are written to main memory towards the minimum of only on replacement. The fourth removes the restriction that when there is one writer of a block that there are no other readers. Each of these enhancements is presented below as a change to the Basic protocol. Their qualitative effects on performance are also assessed. In section 3.2.3 we consider the interactions of the enhancements.

Section 3.2.1. Reduced Memory Writes

The three enhancements that reduce memory writes are:

1) A cache raises the *shared* line when supplying a block to another cache. If the *shared* line is not raised (i.e. memory is the supplier), the receiving cache marks the block as ONLY. In this case, the broadcast write (action 2c above) will be needed in fewer cases.

2) The cache supplier on a READ or READ-MOD request does not write its copy to main memory even if its copy is in state WBACK.

3) Instead of a broadcast write, the cache raises the *invalidate* line for one cycle.

Enhancement two implies the some cache's copy must stay in state WBACK so that the updates are not lost. On a READ-MOD the supplier is going to invalidate its copy so the requester must be responsible. On a READ the supplier must be responsible since the requester might be a read-only cache.

Enhancements one and two clearly improve performance since they decrease bus accesses and/or memory writes. The effect of enhancement three is less clear. Raising the *invalidate* line has a lower cost than a main memory write. On the other hand, if the only write to that block is that one write to that one word, then using the *invalidate* line implies that block write-back is needed on replacement. Which approach is better depends on the probability that there is only one write to the block before replacement.

Section 3.2.2. Readers and Writers

The fourth enhancement allows multiple valid cache copies even if a data block is modified. We refer to this as *multiple readers/writers*. The Basic protocol invalidates other copies on the first write. The first write occurs on a READ-MOD, or on a CPU-WRITE to a in-cache copy that has only been read previously. Thus the changes to the Basic protocol are:

4a) On a READ-MOD, the requester broadcasts the updated word, and all valid copies go to (NOT-ONLY, NO-WBACK).

4b) On all CPU-WRITEs to NOT-ONLY blocks, the cache broadcasts the updated word and writes it to main memory. All other caches with valid copies update them. The state of the block in all caches remains (NOT-ONLY, NO-WBACK).

Enhancement 4, alone, changes the Basic protocol to a pure "write-through" protocol. Unless enhancement 4 is combined with enhancement 1, which uses the *shared* line to notify the writing cache that it has the only copy of a block, the performance of enhancement 4 is in most cases lower than the Basic protocol [SMI82]. From now on we only consider enhancement 4 as combined with enhancement 1. The requester on a READ-MOD only broadcasts the updated word if the *shared* line is raised (by the supplier). Each cache then continues to broadcast writes if the *shared* line is raised on the previous broadcast write. Otherwise, the writer goes to block state (ONLY, WBACK).

The advantage of allowing multiple valid copies is that when a cache wants to read or write its copy that copy is always valid. The disadvantage is that all writes to NOT-ONLY blocks have to be broadcast. This slows the writing cache because the bus has to be obtained and main memory has to be free. The caches with the other copies are also slowed because they may have to block local CPU requests to update their copies.

Whether the advantage or disadvantage dominates depends on the access pattern. A good access pattern for multiple readers/writers is when the majority of the caches with valid copies have high frequencies of accessing their copies and those accesses have a fine temporal granularity of interleaving. A bad access pattern is when one cache frequently writes its copy and the other caches very infrequently read or write their copies.

An extension to the fourth enhancement is possible. The idea is to dynamically switch between the multiple readers/writers approach and the invalidation approach depending on the quality of the access pattern. The key is to find a simple, but accurate dynamic measure of access pattern quality. The measure proposed in the literature [RUD84] is whether or not two subsequent writes to the given block are by the same cache. If so, the interleaving of writes is considered not fine enough to merit the multiple readers/one writer approach.

Section 3.2.3. Combinations of Enhancements

The above enhancements are presented, except as noted, as independent changes to the Basic protocol. When enhancements are combined, the changes to the Basic protocol are somewhat different. The most noteworthy difference occurs when the third and fourth enhancements are combined. Instead of the *invalidate* line being raised, a write is broadcast but not written to main memory. Consequently, some cache has to take responsibility for write-back on replacement. We assume the broadcaster takes responsibility by going to state WBACK. The other caches remain in state NO-WBACK.

Section 3.3. Protocols in the Literature

Our goal is to develop a general framework within which the key traits of the protocols that have appeared in the literature can be identified and evaluated. Given that framework, we are able to quantitatively assess the performance of the Basic Protocol and of the various enhancements to it. We do not claim that the protocols in the literature exactly conform to the framework above. We do, however, feel that the correspondence is close enough to be of interest. Our performance models are described in the next section. Here we relate the model to the specific protocols in the literature.

The Basic protocol is essentially the Write-Once protocol [GOO83], except that another cache will supply the block on a READ or READ-MOD request even if it's copy is in state NO-WBACK. The Synapse protocol [FRA84] modifies the Basic protocol by including enhancement 3. The "ownership-based protocol" [KAT85] builds on the Synapse protocol by adding enhancement 2. Papamarcos and Patel's protocol [PAP84] uses enhancements 1 and 3. In addition, their protocol assumes main memory is updated on the same bus transaction as the cache supply, which should give similar performance to enhancement 2. The Dragon proposal [MCC84] combines all four enhancements. Rudolph and Segall's RWB protocol [RUD84] uses enhancements 1, 3, and 4, including the extension to dynamically switch between the invalidation approach and the multiple readers/one writer approach.

Section 4. The Protocol Models

We have created GTPN models to estimate the performance of the Basic protocol and five protocols which include combinations of enhancements described in the section 3.2. Enhancement 1 can be implemented easily if the bus contains the additional control line we have called *shared* (e.g. if we assume Futurebus rather than Multibus). Thus, we have named the protocol which includes enhancement 1 only, the Smart Basic protocol. The four additional protocols we have studied are called +(2), +(3), +(4), and +(2,3,4), where the numbers refer to the enhancements included in addition to Smart Basic. In this section we briefly describe the GTPN models and the workload parameters. The performance estimates obtained by solving the models are presented in Section 5.

We assume the reader is familiar with the Petri net notation. The GTPN is a Petri net which has the following attributes associated with each transition: 1) a deterministic firing duration which may be marking-dependent, 2) a firing frequency which may be marking-dependent and which determines the probability that the transition will fire, 3) a special flag used to calculate next-state probabilities, and 4) a named set of resources which are "in use" when the transition is firing. For more details on the GTPN and its solution, the reader is referred to [HOL85a].

Section 4.1. The Basic Protocol: Net

The net for the Basic protocol is shown in Figure 4.1. Each token in place P1 represents a processor that has just completed an instruction cycle. We assume that all processors are stochastically homogeneous (i.e. their memory access behaviors are statistically identical). The tokens in places P2 and P9 indicate that the bus and memory are available, respectively.

Each processor token acts independently. During each cycle, the processor continues instruction execution (T1), makes a cache request that can be serviced locally (T3), or makes a cache request that requires the bus (T2). Requests that are serviced locally may be blocked if the cache is servicing a bus request from or supplying data to another cache (T6 or T7). CPU requests that require the bus are either broadcast writes (T9, T13, and T17) or read (i.e. READ or READ-MOD) requests (T8). Read requests are either supplied by another cache (T11) or by main memory (T16). If supplied by another cache, the cache supplier will write the block back to main memory if it was in state WBACK (T18 and T21). After the cache supplier is through, the requesting cache may need to write back a replaced block (T22 and T21). Finally, the CPU request is serviced by the local cache (T5 and T23).

The attributes associated with the transitions in the Basic Protocol model are given in Table 4.1. The reader is encouraged to check that the durations associated with each transition make sense for the model description above. For example, transition T8, which represents a LOAD request on the bus, has a duration of 1 cycle. Transition T10 has a duration of 0 because it is inhibited by certain net markings which will be discussed below.

The frequency expressions are used with the CntCombs flag to calculate next-state probabilities in the Markov Chain. The reader is referred to [HOL85a] for more details. For the purposes of the present discussion, the frequency expressions define relative probabilities of branches in the net.

The frequency expressions may be marking-dependent. That is, they may contain the names of places and/or transitions, which evaluate to the marking of the place or transition in a given state. For example, the frequency of transition T6 evaluates to zero, unless T8 or T13 is firing (i.e. some other cache has a signal on the bus). Similarly, transition T10 is inhibited if T14 or T18 is firing or if P13 contains a token (i.e. a cache request is being supplied by another cache). This should also make sense from the model description above. The probabilities named in the frequency expressions are derived from basic workload parameters, such as the probability that a CPU request is a read (R) or write (W) for a private (P) or shared (S) block. The parameters and the derivation of the frequency expressions are discussed in the next section on the model workload.

The resources associated with the net transitions identify performance measures to be calculated. The long run expected number of usages of each resource is calculated automatically during analysis of the model. We are interested in three measures: bus utilization (Bus), processing power (PP), and speedup (Spd). Each of these resources is "in use" when any of the associated transitions are firing. Bus utilization is straightforward. It is the long run expected fraction of time that T8, T11, T13, T16, T18, or T22 is firing. Both processing power and our speedup measure [HOL85b] can be defined as the average fraction of time a processor is productive times the number of processors (for the homogeneous case). The difference is that processing power only counts the cycles the processor spends executing (T1) as productive, whereas our speedup includes the one cycle required for each cache request (T5 and T23). Processing power is most often used in the performance modeling literature, and is included in our model to show how it is specified. The speedup measure compares the effective computing power of the multiprocessor with a uniprocessor that has an infinite cache. We feel that speedup is a better measure of the perceived performance of memory intensive computations on the multiprocessor.

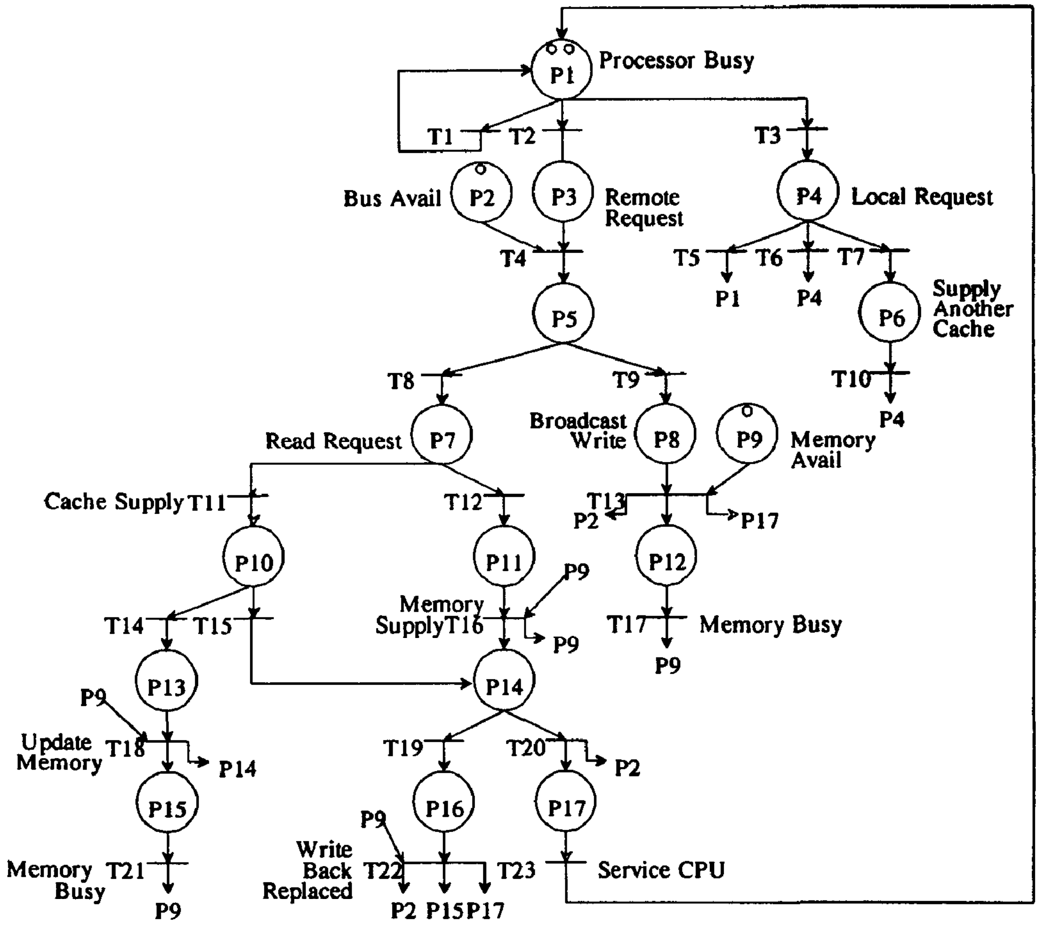

Figure 4.1. GTPN Model for the Basic Protocol

Table 4.1. The Attributes of Basic Protocol Model Transitions

Transition	Duration	Frequency	Cnt Combs	Resources
T1	1.0	1 - ProcReq	yes	(Spd,PP)
T2	0.0	(PSRWM + PSWHumod) * ProcReq	yes	()
T3	0.0	(PSRH + PSWHamod) * ProcReq	yes	()
T4	0.0	1.0	no	()
T5	1.0	1 - Freq(T6) - Freq(T7)	yes	(Spd)
T6	1.0	0.5 * (SRMiss + SWMiss) * T8	yes	()
		+ 0.5 * (ShRead + ShWrite) * T13		
T7	0.0	1/#processors * (T14|P13|T18)	yes	()
T8	1.0	PSRWM / (PSRWM + PSWHumod)	no	(bus)
T9	0.0	1 - Freq(T8)	no	()
T10	0.0	1 - (T14|P13|T18)	no	()
T11	*blocksize*	CSupSR * SRMiss + CSupSW * SWMiss	no	(bus)
T12	0.0	1 - Freq(T11)	no	()
T13	1.0	1.0	no	(bus)
T14	0.0	WBCSupSW * SWCSup	no	()
T15	0.0	1 - Freq(T14)	no	()
T16	*3 + blocksize*	1.0	no	(bus)
T17	1.0	1.0	no	()
T18	*blocksize*	1.0	no	(bus)
T19	0.0	RepP * Priv + RepSW * ShWrite	no	()
T20	0.0	1 - Freq(T19)	no	()
T21	max(4 - *blocksize*, 0)	1.0	no	()
T22	*blocksize*	1.0	no	(bus)
T23	1.0	1.0	no	(Spd)

Two approximations are made in the Basic protocol model. First, the amount of time memory is held for a block write (T18 and T21, or T22 and T21) is sufficient for the first module to finish the write operation. The other blocksize-1 modules will still be busy for up to blocksize-1 cycles into the future. The model is accurate if the next bus access is a load request, but will be slightly inaccurate if a (one-word) broadcast write occurs to one of the modules that is still busy. Second, the amount of time memory is held for a broadcast write (T13 and T17) is approximated to be 2 cycles. The module written to is busy for 4 cycles, but this module may not the the module addressed on the next memory access. Both approximations are necessary because we are representing blocksize memory modules with a single token in place P9. We have verified that the model results are not sensitive to these approximations.

Section 4.2. The Basic Protocol Workload

The performance of the Basic protocol, and the relative improvement in performance for each of the protocol enhancements, is highly dependent on the amount of data sharing that occurs dynamically in the workload. For example, if there is no data sharing, then enhancement 1 might yield some performance improvement over the Basic protocol, but enhancements 2 and 4 will have no effect. Thus, the workload model and parameter values are important considerations.

In the absence of experimental data for multiprocessor workloads, we have based our model on the workload model proposed by Dubois and Briggs [DUB82]. They define the workload for an MIMD machine to be the merge of two memory access streams: one stream for private and shared read-only blocks, and one stream for shared writeable blocks. We have separated the first stream into two streams, one for private, and one for shared read-only blocks. This allows us to define more fundamental workload parameters for our more detailed protocol model. Thus, we view the stream of memory requests in our model to be the merge of three streams, for: 1) private blocks, 2) shared read-only blocks, and 3) shared writeable blocks.

The fundamental parameters for this workload model are shown in Table 4.2. "ProcReq" is the probability that a processor makes a memory request in a given cycle (see transitions T1-T3 in Table 4.1). We assign interrequest times to be geometrically distributed with mean 2.5. This is based on the assumption that a large fraction of interrequest times will be in the range of 0-2 cycles, but several instructions which occur occasionally, such as multiply, can be much longer.

We consider three levels of data sharing: 1%, 5%, and 20%. The probabilities that a memory request is for private (P), shared read-only (SR), and shared writeable (SW) data are shown for each of these cases in the table. The 1% sharing case has probabilities such as might occur if only the operating system shares data. The 20% case has probabilities indicating a tightly-coupled parallel computation that frequently accesses shared read-only data, and thus has good potential for speedup on a multiprocessor. The 5% sharing is an intermediate case.

The parameters for private blocks were chosen according to data reported from extensive uniprocessor cache simulations by Smith [SMI85a]. This includes the hit ratio (HitP), read ratio (ReadP), and Smart Basic probability of write back on replacement (SmartRepP). The RepP parameter for the Basic protocol is somewhat lower because private data is written through to memory on the first write. The hit ratio for SR blocks is assumed to be the same as for P blocks, as in the Dubois and Briggs model. Hit ratio and read ratio for SW data were chosen to be pessimistic estimates which should show maximum performances differences among the different protocols.

Table 4.2. Fundamental Workload Parameters

Parameter	Meaning	Value
ProcReq	Processor request	0.286
Priv, ShRead, ShWrite	processor request to a P, SR, SW block	0.99, 0.00, 0.01 0.95, 0.03, 0.02 0.80, 0.15, 0.05
HitP, HitSR, HitSW	hit given P, SR, SW	0.95, 0.95, 0.5
ReadP, ReadSW	read given P, SW	0.7, 0.5
AmodPWH, AmodSWH	block copy is already modified given PWH,SWH	0.7, 0.3
CSupSR, CSupSW	cache supplier given SR, SW	0.95, 0.5
WBCSupSW	cache supplier write back given SW	0.3
RepP, RepSW	write back replaced block given P, SW	0.2, 0.5
SmartRepP	RepP given Smart Basic	0.3
+2RepSW, +23RepSW	RepSW given +(2), +(2,3)	0.6, 0.7
MrwHitSW	HitSW given multiple readers/writer	0.95

Table 4.3. Intermediate Workload Values

Probability	Derivation
PWH	Priv × (1-ReadP) × HitP
SWH	ShWrite × (1-ReadSW) × HitSW
PSRH	Priv × ReadP × HitP + ShRead × HitSR + ShWrite × ReadSW × HitSW
PSRWM	Priv × (1 - HitP) + ShRead × (1 - HitSR) + ShWrite × (1 - HitSW)
PSWHamod	AmodPWH × PWH + AmodSWH × SWH
PSWHumod	(1 - AModPWH) × PWH + (1 - AModSWH) × SWH
Denom	ShRead × (1 - HitSR) + ShWrite × (1 - HitSW) + Priv × (1 - HitP)
SRMiss	(ShRead × (1 - HitSR)) / Denom
SWMiss	(ShWrite × (1 - HitSW)) / Denom
SWCSup	(ShWrite × (1 - HitSW)) / (ShWrite × (1 - HitSW) + ShRead × (1 - HitSR))

The probabilities that another cache will supply blocks on a miss for SR or SW data, are set equal to the hit ratios. This assumes a high temporal degree of sharing. The remaining workload parameters for SW data were also chosen to be conservative.

The derivation of probabilities used in the frequency expressions of Table 4.1, from the fundamental workload parameters, is given in Table 4.3. For example, a memory request can be served by the local cache (transition T3) if it is a read hit for a private or shared block (PSRH), or if it is a write hit for an already modified block (PSWHamod), as derived in Table 4.3. The interested reader should be able to follow the remaining derivations. Note, in general, "H" stands for hit, "M" stands for miss, and "umod" stands for unmodified in the tables. "SRMiss" should be read as "Shared Read given Miss".

Section 4.3. The Other Protocols

The Smart Basic protocol requires no change in the net, but requires three changes in the transition frequencies. A write-hit to an unmodified private block no longer needs the bus, since the cache will always know it has the only copy of a private block. Thus, the PWHumod term moves from transition T2 to transition T3. This will increase the probability that a private block has to be written back on replacement (RepP becomes SmartRepP for T19). Furthermore, the second term in the frequency expression for T6 changes to 0.5 × T13, since all broadcast writes are to shared blocks. These changes are also made for the remaining protocols. Note that a write-hit to an unmodified shared block (SWHumod) will also not require the bus if the cache has the only copy of the shared block. The probability of this is difficult to estimate, so we choose the conservative approach of assuming the probability is negligible.

Protocol +(2) requires one change in the net, and a further change in the transition frequencies. The column headed by transition T14 is removed and RepSW increases to +RepSW (T19), since the cache supplier does not write back.

The +(3) protocol also requires a change in the net. The place P9 is removed as an input and the column headed by P12 is removed as the output of transition T13, since the invalidation signal does not require a main memory write. Also, +RepSW is used instead of RepSW, since blocks are only written back on replacement. Note that we estimate that the effect of enhancement 3 on RepSW is comparable to the effect of enhancement 2.

Enhancement 4 requires a change in one firing duration, and several changes to the transition frequencies. A requester must broadcast the updated word if a cache supplies the block for a READ-MOD request. Thus, the firing duration of T15 becomes 1.0. Since all write hits to shared blocks are broadcast, SWHamod moves from T3 to T2, and SWH replaces SWHumod in transition T8. Second, HitSW is increased to MrwHitSW, because the shared blocks remain valid.

The +(2,3,4) protocol requires all of the above changes, and uses the ++RepSW parameter instead of RepSW because both 2 and 3 are incorporated.

Section 5. Protocol Performance

Using our GTPN model, we obtained a processing power estimate of 4.1 for the +(2,3) protocol, with 5% sharing and nine processors. This value agrees well with Papamarcos and Patel's approximate analysis [PAP84] for a block transfer time of 4 and similar workload. At 4 processors the bus is only 50% utilized for the +(2,3) protocol, and we find approximately a 10% increase in bus utilization for the Basic protocol. This agrees with the trace simulation results in [KAT85] for 8 kilobyte cache size. These results indicate that the model produces reasonable performance estimates.

The goal of our experiments is to compare the performance of the six protocols (Basic, Smart Basic, +(2), +(3), +(4), and +(2,3,4)), given reasonable values for architectural and workload parameters. Two architectural parameters considered are the *blocksize* and the main memory cycle time. Both are four in our initial experiments. The performance measure primarily used is *speedup*, as defined in Section 4.1.

Section 5.1. Effect of Enhancements

Figure 5.1. shows the results of our initial experiments. The curves start below one on the y-axis because we have analyzed the uniprocessor with cache misses (but with frequency 0 for T6, T7, and T11). The bottom dashed, dotted, and solid curves are for the Basic protocol. The next set of curves are the speedup for the Smart Basic protocol. The curve for Smart Basic with 20% sharing is almost hidden by the curve for Basic with 1% sharing. The top three curves are the speedup for the +(4) protocol.

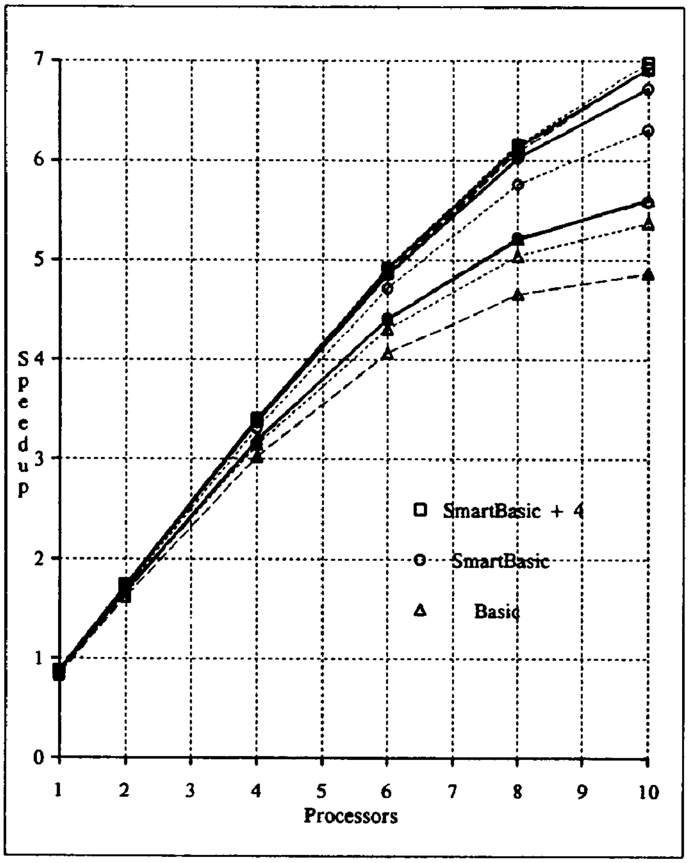

Figure 5.1. Comparison of Protocols for 1% (solid), 5% (dotted), and 20% (dashed) Sharing

The curves for the +(2) and +(3) protocols are nearly identical to the Smart Basic protocol. The speedup for +(2) is at most 2.5% greater than Smart Basic. The speedup for +(3) is at most 0.1% less than Smart Basic. For the sake of readability, these curves are not shown in the figure. Similarly, the +(2,3,4) protocol curves are not drawn because they are indiscernable from the +(4) protocol. Table 5.1 summarizes the speed-up estimates for all of the protocols for the 10-processor case.

Clearly, adding enhancement one substantially improves system performance. Enhancements two and three have negligible effect when added to Smart Basic. In fact, for the workload parameters used, the effect of enhancement 3 is very slightly negative. Enhancement 4 shows a small improvement over Smart Basic for our workload with 1% sharing, but shows more substantial improvement as sharing is increased to 5% and 20%.

Table 5.1. Summary of Speed-Up Estimates for 10 Processors

Protocol	Percent Sharing		
	1%	5%	20%
Basic	5.602	5.371	4.868
SmartBasic	6.718	6.310	5.582
+(2)	6.746	6.370	5.715
+(3)	6.716	6.306	5.576
+(4)	6.913	6.983	6.929
+(2,3,4)	6.911	6.993	6.972

The spread between the solid, dotted, and dashed lines indicates that the amount of sharing has a significant effect on the performance of the Basic, Smart Basic, +(2), and +(3) protocols. This is reasonable because the number of broadcast writes and the average hit rate are proportional and inversely proportional, respectively, to the amount of sharing in these protocols. In contrast, the amount of sharing has little influence when enhancement 4 is included. This is because the hit rate for shared data was assumed to be high for enhancement 4.

The effectiveness of enhancement 4 in Figure 5.1 requires further study because our model assumed a good access pattern for multiple readers and writers (see section 3.2.2). To model a bad access pattern we reran +(4) at 20% sharing without increasing the hit rate. The results were very slightly lower than the Smart Basic results, because there are more broadcast writes. Thus, enhancement 4 helps significantly (especially at large amounts of sharing) when there is a good access pattern and has negligible effect when there is a bad access pattern. It appears that dynamically switching between this and the invalidation approach (enhancement 3) offers no advantages.

The curves in Figure 5.1 terminate at ten processors because of rapid growth in the size of the state space. Table 5.2 shows this growth for the Basic Protocol. Nonetheless, none of the curves will rise much further because of bus saturation. Bus utilizations vary from 89% to 98% at ten processors for the workloads and protocols in Figure 5.1. In section 5.3 we show that the assumed hit rates in our initial workload are primarily responsible for the bus saturation at 9-10 processors.

Table 5.2. State Space Growth in the Basic Protocol

Processors	States
1	33
2	355
4	2,364
6	7,961
8	19,856
10	41,159

Section 5.2. Effect of Blocksize

Figure 5.2 shows our experiments which varied the blocksize. The experiments in Figure 5.1 assumed a blocksize of four words. Blocksizes of one and eight are considered here, for the Smart Basic protocol. The one word *blocksize* case requires a change in the protocol. In this case, there is no need to load a block on a write miss, since whatever value is loaded is completely overwritten. However, the other caches still must be notified. Thus, a write miss can be implemented as a broadcast write. The net does not have to be changed. The write miss probability (PSWM), however, is moved from transition T8 to transition T9.

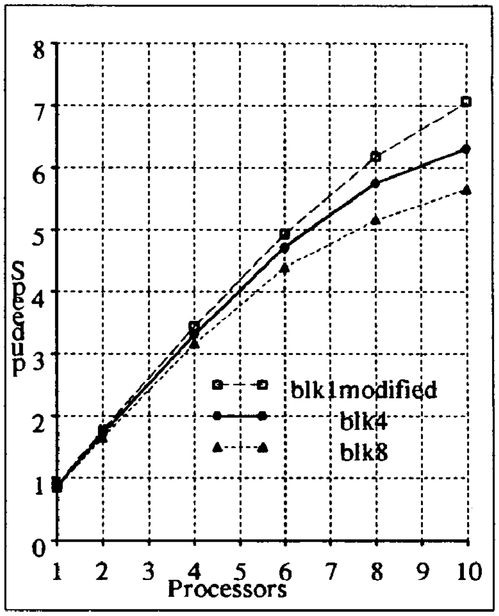

Figure 5.2. Varying Block Size. SmartBasic at 5% sharing.

The key effect of changing blocksize on the workload is on the hit rate. Based on Alan Smith's results for an 8 kilobyte cache [SMI85b], we reduce HitP and HitSR from 95% to 90% for a blocksize of one, and increase these values to 97% for a blocksize of eight. HitSW is reduced to 40% for blocksize one, and increased to 55% for blocksize eight, compared with 50% for blocksize four. Other parameters, such as AmodPWH, AmodSWH, WBCSupSW, RepP, and RepSW, were also suitably modified.

Blocksize of one, with the protocol appropriately modified, performs best in these experiments. However, smaller blocksizes have substantial disadvantages due to a larger number of memory accesses during task switching, which we have not modeled. A study of blocksize effects which includes task switching in the workload would be worthwhile.

Section 5.3. Effect of Hit Rate and Memory Speed

Figure 5.1 indicates that at most ten or so processors should be put on a single bus. These results are likely to be very conservative because we have chosen many of our parameter values very conservatively. Two parameter changes seem especially worthwhile to consider. The first change increases the hit rate for private and SR blocks from 95% to 99%. The second change assumes a faster main memory. In particular, we investigate a shared memory cycle time of two instead of four. Note that main memory accesses still require a minimum of 4-5 cycles, due to acquiring the bus and memory, transmitting the bus request, and servicing the CPU when the request is satisfied. The resulting four curves are shown in Figure 5.3 for the Smart Basic protocol at 20% sharing.

Though both changes are significant, the hit rate increase is clearly more important. This is also demonstrated when bus utilization is considered. At nine processors the bus utilization for the original model is 96%. The utilization for the model with main memory cycle time of two and 95% hit rate is 90%. The utilization for the model with main memory cycle time of two and 99% hit rate is 53%. Thus, the faster memory probably only allows one or two more processors. The higher hit rate should allow several more processors, perhaps six or eight, to be added. Recall that this is for the 20% sharing level. Further increases in the number of processors that can be supported can be expected for the reduced sharing workloads.

Section 6. Conclusion

We have used an exact analytic technique, based on Generalized Timed Petri Nets, to derive performance estimates for shared bus cache consistency protocols. Using the GTPN model,

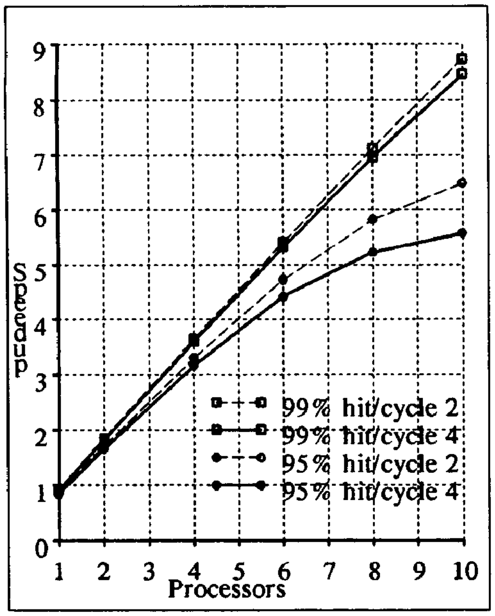

Figure 5.3. Effect of Hit Ratio and Main Memory Cycle Time. Smart-Basic at 20% sharing.

we were able to specify both constant delays and instantaneous events for detailed bus and memory activity, which must be represented to evaluate these protocols. The GTPN can be solved for steady-state performance estimates with these parameters, in contrast to previous Stochastic Petri Net models. Performance estimates were obtained automatically using Markov Chain techniques.

The workload in these studies was based on the model used by Dubois and Briggs [DUB82], but included separate specification of private and shared read-only data access behavior. We considered three levels of data sharing in our workloads: 1%, 5%, and 20%. Parameter values for private data were based on uniprocessor cache simulations by Smith [SMI85a]. We found that the level of sharing, and the assumed cache hit rate for private and shared read-only data, have a substantial impact on the performance estimates.

We evaluated four enhancements to the Write-Once protocol which have appeared in the literature. One enhancement makes use of the *shared* bus line which is raised by a cache when supplying another cache, as proposed in [PAP84, MCC84, and RUD84]. This enhancement improves system performance by as much as 15-20% at all levels of data sharing studied. Another enhancement (enhancement 4 in our models) allows multiple readers and writers for a data block, as proposed in [MCC84 and RUD84]. This enhancement further improves performance by as much as 10-20% for the 5% and 20% sharing workloads, as long as there is a fine temporal interleaving of accesses to the data block by more than one cache. Also, this enhancement does not decrease performance appreciably in other workloads we studied. The remaining two enhancements which have been proposed in the literature do not show significant performance improvement for any of the workloads studied.

The GTPN model is concise and precise, and can be used to derive estimates for additional workload parameters of interest to system designers, or as experimental data for multiprocessors becomes available. The major disadvantage of the GTPN model appears to be the rapid growth in the state spaces. Consequently, future research plans also include investigating approximate techniques for solving GTPN models with larger state spaces.

Acknowledgements

We would like to thank Jim Goodman and Andrew Pleszkun for many useful discussions and the use of their VAX/750 throughout our experiments, and Jim Archibald and Jean-Loup Baer for an early copy of their cache coherency protocol simulation results.

References

[AJM84] Ajmone Marsan, M., G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets", *ACM Trans. on Computer Systems*, Vol. 2, May 1984, pp. 93-122.

[ARC85] Archibald, J., and J.-L. Baer, "An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors," Technical Report 85-10-05, Dept. of Comp. Sci., Univ. of Washington, October 1985.

[DUB82] Dubois, M., and F. A. Briggs, "Effects of Cache Coherency in Multiprocessors", *IEEE Trans. on Computers*, Vol. C-31, November 1982, pp. 1083-1099.

[FRA84] S.J. Frank, "Tightly Coupled Multiprocessor System Speeds Memory Access Times," *Electronics*, Vol. 57, no. 1, January 1984, pp. 164-169.

[GOO83] Goodman, J.R., "Using Cache Memory to Reduce Processor-Memory Traffic," in *Proc. of 10th Int. Symp. on Computer Architecture*, June 1983, pp. 124-131.

[HOL85a] Holliday, M. A., and M. K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," *Proc. Int'l. Workshop on Timed Petri Nets*, Torino, Italy, July 1985.

[HOL85b] Holliday, M. A., and M. K. Vernon, "Exact Performance Estimates for Multiprocessor Memory and Bus Interference", Technical Report #594, Comp. Sci. Dept., UW-Madison, May 1985.

[KAT85] Katz, R., S. Eggers, D.A. Wood, C. Perkins, and R.G. Sheldon, "Implementing a Cache Consistency Protocol," *Proc. of 12th Int. Symp. on Computer Architecture,* June 1985, pp. 276-283.

[MCC84] McCreight, E., "The DRAGON Computer System: An Early Overview," *NATO Advanced Study Institute on Microarchitecture of VLSI Computers,* Urbino, Italy, July 1984.

[MOL82] Molloy, M. K., "Performance Analysis Using Stochastic Petri Nets", *IEEE Trans. on Computers*, Vol. C-31, Sept. 1982, pp. 913-917.

[MOL85] Molloy, M. K., "Discrete-Time Stochastic Petri Nets", *IEEE Trans. on Soft. Engr.*, Vol. SE-11, April 1985, pp. 417-423.

[NAT80] Natkin, S., "Reseaux de Petri Stochastiques", These de Docteur-Ingenieur, CNAM-Paris, June 1980.

[PAP84] Papamarcos, M., and J. Patel, "A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories," *Proc. of 11th Int. Symp. on Computer Architecture,* June 1984, pp. 348-354.

[RUD84] Rudolph, L., and Z. Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," *Proc. of 11th Int. Symp. on Computer Architecture,* June 1984, pp. 340-347.

[SMI82] Smith, A.J., "Cache Memories," *Computing Surveys*, Vol. 14, no. 3, pp. 473-530, September 1982.

[SMI85a] Smith, A. J., "Cache Evaluation and the Impact of Workload Choice", *Proc. of 12th Int. Symp. on Computer Architecture,*, June 1985.

[SMI85b] Smith, A. J., "Line (Block) Size Choice for CPU Cache Memories", Technical Report CSD 85/239, Computer Science Division, Univ. of Calif. at Berkeley, 1985.