An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols †

M. K. Vernon
E. D. Lazowska
J. Zahorjan

Computer Sciences Technical Report #746

February 1988

† To appear in *Proc. 15th Ann. Int'l. Symp. on Computer Architecture,* Honolulu, Hawaii, May 30 - June 2, 1988.

An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols

M. K. Vernon

Department of Computer Science
University of Wisconsin-Madison
Madison, WI 53706

E. D. Lazowska
J. Zahorjan

Department of Computer Science
University of Washington
Seattle, WA 98195

Abstract

A number of dynamic cache consistency protocols have been developed for multiprocessors having a shared bus interconnect between processors and shared memory. The relative performance of these protocols has been studied extensively using simulation and detailed analytical models based on Markov chain techniques. Both of these approaches use relatively detailed models, which capture cache and bus interference rather precisely, but which are highly expensive to evaluate. In this paper, we investigate the use of a more abstract and significantly more efficient analytical model for evaluating the relative performance of cache consistency protocols. The model includes bus interference, cache interference, and main memory interference, but represents the interactions between the caches by steady-state mean collision rates which are computed by iterative solution of the model equations.

We show that the speedup estimates obtained from the mean-value model are highly accurate. The results agree with the speedup estimates of the detailed analytical models to within 3%, over all modifications studied and over a wide range of parameter values. This result is surprising, given that the distinctions among the protocols are quite subtle. The validation experiments include sets of reasonable values of the workload parameters, as well as sets of unrealistic values for which one might expect the mean-value equations to break down. The conclusion we can draw is that this modeling technique shows promise for evaluating architectural tradeoffs at a much more detailed level than was previously thought possible. We also discuss the relationship between results of the analytical models and the results of independent evaluations of the protocols using simulation.

1. Introduction

High-speed local memory that operates as a cache for a larger or more distant main memory can significantly increase the effective execution rate of a processor. When this technique is used in a general-purpose shared-memory MIMD multiprocessor, the problem of maintaining consistency among the data stored in multiple caches arises.

A number of dynamic cache consistency protocols have been developed for the case where the multiprocessor interconnection network is a shared bus. In this case, each cache controller monitors the traffic on the bus, and takes appropriate actions to maintain data consistency. The simplest protocol is a write-through protocol, in which all writes to a cache are written through to main memory, causing other caches that have the data to update or

Our program of research in computer system performance analysis is supported by the National Science Foundation (Grants No. DCR-8352098, DCR-8451405, CCR-8619663, and CCR-8703049), the Naval Ocean Systems Center, U S WEST Advanced Technologies, the Washington Technology Center, Digital Equipment Corporation (the Systems Research Center and the External Research Program), CRAY Research, the AT&T Foundation, Bell Communications Research, Boeing Computer Services, Tektronix, Inc., the Xerox Corporation, and the Weyerhauser Company.

invalidate their copies [Smit82]. In 1983, Goodman proposed a copy-back multi-cache consistency protocol that has since become known as the Write-Once protocol [Good83]. Since that time, a number of more sophisticated protocols have been proposed. These include the Synapse protocol [Fran84], the Illinois protocol [PaPa84], the RWB protocol [RuSe84], the Dragon protocol [MCCr84], and the Berkeley protocol [KEWP85].

The relative performance of the above protocols has been studied extensively using simulation [ArBa86]. In another approach, the key modifications to the Write-Once protocol that have been included in each of the five successor protocols were identified, and the contribution of each to overall performance was evaluated [VeHo86]. This second study used an analytic technique called Generalized Timed Petri Nets (GTPNs) [HoVe85]. (Results of the GTPN model agreed very well with results of independent evaluations in the various protocol proposals.)

Both of the above studies used relatively detailed models, which capture cache and bus interference rather precisely, but which are highly expensive to evaluate. In this paper, we investigate the use of a more abstract and significantly more efficient analytical model for evaluating the relative performance of the cache consistency protocols. We use the abstract model to reproduce and extend the results in [VeHo86]. The model includes bus interference, cache interference, and main memory interference, but represents the interactions between the caches by steady-state mean collision rates which are computed by iterative solution of the model equations.

One might expect the more abstract mean-value model to be less accurate than the detailed GTPN model. However, we show that the speedup estimates obtained from the mean-value model are surprisingly accurate. The results agree with the speedup estimates of the GTPN model to within 3%, over all modifications studied and over a wide range of parameter values. The validation experiments include sets of reasonable values of the workload parameters, as well as sets of unrealistic parameter values for which one might expect the mean-value equations to break down. The conclusion we can draw is that this modeling technique shows promise for evaluating architectural tradeoffs at a much more detailed level than was previously thought possible.

Greenberg and Mitrani have also recently developed an analytical model of the Write-Through, Write-Once, and Dragon protocols [GrMi87]. Our mean-value model of cache, main memory, and bus interference is more comprehensive and more firmly grounded in queueing network theory than their model. Furthermore, we are able to consider deterministic bus access times for cache block transfers, rather than the exponential access times required in their model. However, our workload model is less sophisticated than their workload model and the workload model in [ArBa86]. We comment on this further in Section 2.3.

The remainder of this paper is organized as follows. Section 2 contains a brief review of the Write-Once protocol, the four modifications to Write-Once that have been proposed, and the workload model in [VeHo86], which is also used in this paper. Section 3 presents our mean-value analytical model of Write-Once and the protocol modifications. Section 4 compares the estimates obtained using the mean-value model with estimates obtained using the GTPN model, and with estimates published in independent studies, for a variety of parameter settings. This section also contains some new results, including the asymptotic performance of the modifications. Section 5 contains the conclusions of this work.

2. Background

The multiprocessor configuration assumed for the snooping cache consistency protocols is illustrated in Figure 2.1. Each processor is connected directly to its local cache. Each cache is connected through the shared bus to main memory and to all other caches.

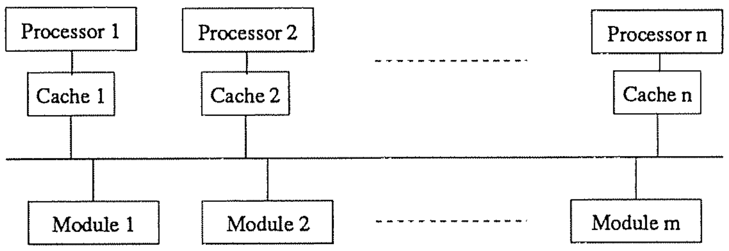

Figure 2.1: Multiprocessor Configuration

2.1. System Assumptions

The system assumptions made in the model developed in this paper are those in [VeHo86]. These assumptions are very similar to the assumptions in [ArBa86], and include the following.

A processor executes for a variable number of cycles, assumed to be exponentially distributed with mean τ, between memory requests. Useful execution is not overlapped with fetching data from memory. Thus, once the request is made, the processor is idle until the request is satisfied. The cache takes one unit of time to satisfy the processor request, either immediately, or following a required bus transaction.

Bus transactions may be one of five types: *read*, *read-mod* (i.e., read-with-the-intent-to-modify), *invalidate*, *write-word*, or *write-block*. The first and second request types are issued when processor read and write requests miss in the cache, respectively. The third or fourth request type is used by the consistency protocol when a processor write request hits in the cache, and the cache block is clean (i.e., unmodified). The fifth request type is used to write a modified data block back to main memory.

Bus requests are served in random order in the GTPN model [VeHo86], but are assumed to be scheduled in first-come first-served order in the mean-value model developed in this paper. Both scheduling disciplines have the same mean waiting time, and thus yield the same predicted speedup measures. Bus requests have priority over processor requests for service in a cache. Dual directories are assumed, so processor requests are only delayed by bus requests that require some action on the part of the cache.

The main memory is divided into m modules, where m is the cache block size, assumed to be four in this paper. Main memory latency is assumed to be three cycles.

Cache block states are assumed to be defined by three bits of state information. (Not all bits are used in exactly the same way by each protocol.) The first bit denotes whether the block is *valid* or *invalid.* The second bit indicates whether the cache knows that it has the only copy of a block, i.e., state *exclusive*, or does not know that it has an exclusive copy, i.e., state *non-exclusive*. The third bit (*wback/no-wback*) denotes whether or not the processor must write back the block when it is purged from the cache. Note that this third bit indicates whether or not the block is modified relative to main memory.

2.2. Review of Snooping Cache Protocols

Below we review briefly the Write-Once protocol, and each of the key modifications that have been proposed to improve performance. In some cases we comment on the potential advantages and disadvantages of the proposed modifications. The reader is referred to [ArBa86], [VeHo86], and the original papers for further detail.

Write-Once

A bus read request loads the cache block in state *non-exclusive* and *no-wback*. A bus read-mod request invalidates all other copies of the block, and loads the block in state *exclusive* and *wback*.

The key idea in the Write-Once cache consistency algorithm is that the *first* time a processor writes a word to a *non-exclusive* block in its cache, the word is written through to main memory. When the word is broadcast on the bus, any cache containing the block invalidates its copy. The write operation changes the state of the block to *exclusive* and *no-wback*.

Writes to a block in state *exclusive* in the cache are written only locally, changing the state to *wback*. If another cache requests the block, indicated by a read or read-mod operation on the bus, a cache containing the block in state *wback* interrupts the bus transaction and writes the block to main memory, thereby updating the contents of main memory before main memory supplies the requested data. The state of the block changes to *no-wback* if the bus request is of type read.

In this protocol, if a cache contains a block in state *wback*, it is the only cache containing the block.

Modification 1

In the first of the proposed modifications, a cache containing a copy of a block requested by a read or read-mod bus operation must raise a *shared* line on the bus. If this line is not raised, the cache block can be loaded in state *exclusive* in the requesting cache. Writes to this block by the requesting cache will not require bus operations to notify other caches. (However, note that writing the block to main memory will be required when the block is purged from the cache.)

This modification reduces the total number of bus operations required by an amount that depends on 1) the workload characteristics and 2) which, if any, of the other three modifications are also implemented. If this is the only modification to the Write-Once protocol, the number of operations required is reduced in the following case: 1) a requested block is not resident in another cache, and 2) the block is written more than once during its tenure in the cache. Modification 1 is included in the Illinois, Dragon, and RWD protocols.

Modification 2

In the second of the proposed modifications, a cache that has a requested block in state *wback*, supplies the copy directly to the requesting cache and does not update main memory. This saves memory traffic and bus operations in the case that the requesting cache modifies the block before purging it.

If the bus operation is a read request, the supplying cache takes responsibility (sometimes called *ownership*) for writing back the block when it is purged. In other words, the supplying cache sets the state to *non-exclusive* and *wback*, and the requesting cache sets the state to *non-exclusive* and *no-wback*, if the bus request is a read operation. Modification 2 is included in the Berkeley and Dragon protocols. The Illinois protocol assumes the data is written to memory and supplied to the requesting cache in the same bus operation, which is another optimization similar to this modification.

Modification 3

In the third modification to Write-Once, a bus *invalidate* operation is performed, instead of the write-word operation, on the first write to a non-exclusive data block. This potentially reduces memory traffic, and eliminates the need for "partial write" operations on the bus. There is potentially a reduction in bus traffic in the case that *write-word* requires two bus cycles and *invalidate* requires one cycle. On the other hand, there is the potential for increased bus traffic with this modification, since the write-word operation sometimes takes the place of writing the entire block to main memory when the block is purged. Modification 3 is included in all five protocols proposed as improvements to Write-Once.

Modification 4

The final modification allows multiple copies of a cache block to remain valid even in the presence of write operations. In this case, all writes to a block in state *non-exclusive*, are broadcast on the bus. All caches update their copies, and main memory is updated by the broadcast write. Cache blocks remain in state *no-wback*.

Note that this modification alone reduces the Write-Once protocol to a write-through protocol. Thus, this modification is only practical when implemented together with modification 1. Modification 4 is included in the RWB and Dragon protocols. Furthermore, the RWB protocol includes the capability to switch between invalidation and broadcast write operations.

Summary

We have presented the above modifications, except as noted, as independent modifications to the Write-Once protocol. The modifications can be implemented in any combination, if the following observation is noted. If modifications 3 and 4 are implemented together, the effect is to broadcast all write operations to non-exclusive blocks, but not to update main memory. In this case, as in modification 2, some cache has to take responsibility for writing back the block when it is purged. We assume the cache performing the broadcast takes this responsibility.

2.3. Our Workload Model

The workload model we use to evaluate the above protocols is the same as the workload model in [VeHo86]. This model was based on the model in [DuBr82], which views the memory reference stream as the merging of two streams, one for private and shared read-only blocks, and one for shared-writable blocks. The first stream was decomposed into two substreams in [VeHo86]. All three streams are treated probabilistically in our model. The probabilistic treatment is very similar to the treatment of the private stream in [ArBa86]. However, less locality is assumed in the shared-writable stream.

The following basic parameters are specified for our workload model:

- $p_{private}$, p_{sro}, and p_{sw}, are the probabilities that a memory reference is to a private, shared read-only (sro), and shared-writable (sw) block, respectively.
- $h_{private}$, h_{sro}, and h_{sw}, are the hit rates for private, sro, and sw streams, respectively.
- $r_{private}$ (r_{sw}) is the probability that the processor request is a read request, given that the reference is of type private (sw).
- $amod_{private}$ ($amod_{sw}$) is the probability that a reference to a private (sw) block that hits in the cache finds the block already modified.
- $csupply_{sro}$ ($csupply_{sw}$) is the probability that a copy of a requested sro (sw) block exists in at least one other cache.
- $wb_{csupply}$ is the probability that the cache supplier contains the data in state *wback*.
- rep_p (rep_{sw}) is the probability that a private (sw) block must be written back to memory when it is purged.

From these parameters, the following model inputs can be computed [VeHo86]:

- p_{local}, the probability that a memory request can be satisfied locally in the cache,
- p_{bc}, the probability that the memory request requires a broadcast write or invalidation,
- p_{rr}, the probability that the memory request requires a remote read or read-mod operation,
- t_{read}, the mean bus access time for a remote read or read-mod operation, which includes main memory write-back by another cache and/or by the requesting cache, if necessary,
- $p_{csupwb|rr}$, the probability that another cache must write the block to main memory in response to a remote read request, and
- $p_{reqwb|rr}$, the probability that the requesting cache must write back a replaced block on a remote read operation.

We note that our probabilistic treatment of the shared data reference stream treats the relationship between system size and *actual* sharing of data more approximately than the workload models in [ArBa86] and [GrMi87]. The workload submodel of both the mean value model in this paper and the GTPN model should be improved to treat the shared references more similarly to the model in [GrMi87]. However, this should not change the conclusions of this paper with regard to the relative accuracy of the mean value model. Furthermore, we show in Section 4 that results of the model agree well with independent evaluations of the protocols, in spite of the approximate workload representation.

3. Mean Value Models of the Cache Consistency Protocols

In this section, we develop a mean-value model of cache, memory, and bus interference for the Write-Once protocol. We then briefly discuss the iterative model solution technique, and the required model modifications for each of the four potential improvements outlined in the previous section. The results of the mean value models are compared with the results of the corresponding GTPN models in Section 4.

The idea in Mean Value Analysis, which has been used with great success to solve queueing network models of computer system performance, is to construct a set of equations that compute the mean values of various performance quantities in terms of the mean values of various model inputs – frequently resorting to iteration when a direct calculation is not possible. What is important to bear in mind (and will be discussed in Section 3.2) is that our mean value approach to analyzing multi-cache consistency protocols yields a dramatic improvement in the time required to obtain performance measures from the model – a reduction from hours of computing to seconds of computing for large numbers of processors – with a negligible loss in accuracy, as will be shown in Section 4.

The notation used in this section is defined in the development of the equations.

3.1. The Write-Once Protocol

We first consider response times for memory requests, then mean bus waiting time, memory interference, and cache interference.

Response Time Equations

The mean total time between memory requests issued by a processor, R, is the sum of the processor execution time between requests, τ, the weighted mean delays for each of the three ways in which memory requests are handled by the cache (locally, broadcast write-word, or remote read), and the cache supply time ($T_{supply} = 1.0$):

$$R = \tau + R_{local} + R_{broadcast} + R_{RemoteRead} + T_{supply}. \quad (1)$$

The weighted mean response time for a memory request that can be handled locally in the cache, R_{local}, is the product of the probability that the request can be handled locally, the mean number of consecutive bus requests that delay the cache response to the processor request ($n_{interference}$), and the mean number of cycles that each interfering bus request requires in the cache ($t_{interference}$):

$$R_{local} = p_{local} \times n_{interference} \times t_{interference}. \quad (2)$$

The calculation of $n_{interference}$ and $t_{interference}$ is discussed in the subsection on cache interference below.

The mean response time for broadcast write operations is estimated as the sum of the mean waiting time for the bus (w_{bus}), the mean waiting time for the main memory module (w_{mem}), and the fixed bus access time for the write-word operation (T_{write} = 1.0). Thus, the weighted mean response time is given by:

$$R_{broadcast} = p_{bc} \; (w_{bus} + w_{mem} + T_{write}). \tag{3}$$

Equations for w_{bus} and w_{mem} are developed in the following subsections on mean bus waiting time and memory interference, respectively.

Finally, the mean response time for a remote read (or read-mod) operation, is approximately the sum of the mean bus waiting time, and the mean bus access time for the read operation (t_{read}):

$$R_{RemoteRead} = p_{rr} \; (w_{bus} + t_{read}). \tag{4}$$

Memory interference is not an important factor in the response time for remote reads. This is due to the fact that main memory latency is assumed to be fixed and small (i.e., 3.0 cycles). Thus, the mean wait for memory after the request is served by the bus, is negligible. Note that t_{read} includes the mean time for one and possibly a second and third cache block transfer on the bus, as defined in Section 2.3.

Mean Bus Waiting Time

To complete the response time calculations in equations (1)-(4), we need to compute w_{bus}, w_{mem}, $n_{interference}$, and $t_{interference}$. The equations for w_{bus} are developed next.

An arriving request will wait for the mean remaining bus access time (i.e., the mean *residual life*, $t_{res,bus}$) of the request in service, plus one mean bus access time (t_{bus}) for every other request in the queue when it arrives. Let $\bar{Q}_{bus}$ represent the mean number requests found in the queue by the arrival, and $p_{busy,bus}$ represent the probability an arriving request finds the bus busy. The equation for w_{bus} is thus:

$$w_{bus} = (\bar{Q}_{bus} - p_{busy,bus}) \; t_{bus} + p_{busy,bus} \; t_{res,bus} . \tag{5}$$

Applying techniques from Product Form queueing networks [LZGS84] in an approximate way, the mean queue length seen by an arriving request is estimated by the steady state mean queue length in the system if the requesting cache were removed. This is approximately the product of the average fraction of time each cache spends in the bus queue, and the number of other caches ($N-1$):

$$\bar{Q}_{bus} = (N-1) \; \frac{R_{bc}+R_{rr}}{R}. \tag{6}$$

Bus utilization is estimated by the product of the number of caches in the system, and the fraction of time each cache uses the bus:

$$U_{bus} = N \times \frac{p_{bc}\,(w_{mem} + T_{write}) + p_{rr} \times t_{read}}{R}, \qquad (7)$$

from which we can estimate the probability that an arriving request finds the bus busy:

$$p_{busy,bus} = \frac{U_{bus} - \frac{U_{bus}}{N}}{1 - \frac{U_{bus}}{N}}. \qquad (8)$$

Finally, the mean bus access time, and mean residual life of the request in service, are given by the following weighted sums:

$$t_{bus} = \frac{p_{bc}}{p_{bc} + p_{rr}}\,(T_{write} + w_{mem}) + \frac{p_{rr}}{p_{bc} + p_{rr}}\,t_{read}, \qquad (9)$$

and

$$\begin{aligned} t_{res,bus} = {} & \frac{p_{bc}\,(T_{write} + w_{mem})}{p_{bc}\,(T_{write} + w_{mem}) + p_{rr}\,t_{read}} \times \frac{T_{write} + w_{mem}}{2} \\ & + \frac{p_{rr}\,t_{read}}{p_{bc}\,(T_{write} + w_{mem}) + p_{rr}\,t_{read}} \times \frac{t_{read}}{2}. \end{aligned} \qquad (10)$$

Memory Interference

The mean time that a broadcast write operation waits for the main memory module (w_{mem}) is the product of the probability that the request finds the module busy ($p_{busy,mem}$) and the mean remaining memory latency. Letting d_{mem} represent the mean total memory latency, assumed to be 3.0 in this paper, the mean memory waiting time is given by:

$$w_{mem} = p_{busy,mem} \times \frac{d_{mem}}{2}. \qquad (11)$$

The probability the request finds the module busy is computed from the utilization of the memory module, in the same way that $p_{busy,bus}$ was estimated from the bus utilization. The utilization of the memory module is estimated as the fraction of time each cache uses the memory module, times the number of caches in the system:

$$U_{mem} = N \times \frac{1}{4} \times \frac{[\,p_{bc} + p_{rr}\,(p_{csupwb|rr} + p_{repwb|rr})\,] \times d_{mem}}{R}. \qquad (12)$$

Note that the above equation assumes four memory modules, but can easily be modified for some other number of modules.

Cache Interference

The cache interference submodel involves computing $n_{interference}$ and $t_{interference}$.

The mean number of consecutive bus requests that interfere with a processor request, $n_{interference}$ is computed assuming, approximately, that the maximum number of requests that can interfere with the processor request is equal to the number of requests in the bus queue when the processor request is issued (i.e., $\bar{Q}_{bus}$).

Using the model input parameters, and assuming that a block supplied by a cache is equally likely to be supplied by any of the other caches, it is straightforward to compute the following probabilities (see Appendix B): 1) the probability, p, that a cache must service a bus request, and 2) the probability, $p' < p$, that the cache must service the bus request for the entire duration of the bus transaction. An example of an event of the second type is a broadcast write operation on a block contained in the cache. An example of an event of the first type, but not of the second type, is a read-mod operation where the cache has the block in state *no-wback*. The cache must respond by invalidating the block, which is of shorter duration that the bus transaction.

$n_{interference}$ is easily computed from p and p', as follows:

$$\begin{aligned} n_{interference} &= \bar{Q}_{bus}\, p'^{\bar{Q}_{bus}-1}\, p + \sum_{k=1}^{\bar{Q}-1} k\, p'^{k-1}[\,(p-p') + p'(1-p)\,] \\ &= p\left(\frac{1-p'^{\bar{Q}}}{1-p'}\right). \end{aligned} \tag{13}$$

$t_{interference}$, the mean cache interference time per request that blocks a processor request, is computed from model inputs in the same way that p and p' are computed (see Appendix B).

3.2. Solution of the Model Equations

The above equations contain cyclic interdependencies, in which R depends on the mean bus and memory waiting times, which in turn depend on R. Thus, the equations must be solved iteratively. We do so, starting with all waiting times set to zero. Solution of the equations converged within 15 iterations in all experiments reported in this paper, yielding results in under one second of cpu time, independent of the size of the system analyzed. In contrast, the time to solve the GTPN model increases exponentially with the number of processors analyzed. With ten processors, the GTPN model requires on the order of one hour of CPU time on a DEC MicroVAX-II with eight megabytes of memory. We note that simulation is equivalently expensive.

3.3. Models of the Four Protocol Modifications

The changes required in the mean-value model for each of the protocol modifications in Section 2.2, are a subset of the required changes in the GTPN model [VeHo86], and mostly involve changes in computing the model inputs.

For modification 1, the calculation of $p_{broadcast}$ no longer includes a term for write hits to private blocks. This term is instead added to p_{local}. The equation for p, the probability of cache interference, must be modified, since write hits to private blocks are no longer broadcast (i.e., p increases slightly). Furthermore, the input parameter rep_p must be increased (from 0.2 to 0.3 in the workload of [VeHo86]), which causes a small increase in t_{read}. For modification 2, the calculations of $t_{contention}$ no longer includes the term for cache supply write-back, and the input parameter rep_{sw} increases (from 0.5 to 0.6 in the workload of [VeHo86]), causing a slight increases in t_{read} and a slight decrease in p'. For modification 3, the term for broadcast writes is removed from equation (12), and the probability rep_{sw} increases, by an amount assumed to be comparable to the effect of modification 2. Finally, for modification 4, changes are required in the calculation of p_{bc}, p_{local}, and p.

4. Results and Accuracy of the Mean Value Models

Speedup is computed from the mean-value models using the formula: $N \times \frac{\tau + T_{supply}}{R}$. The input parameter values specified in [VeHo86] are reproduced in Appendix A. The mean value analysis speedup estimates for these parameter values are plotted in Figure 4.1 and tabulated in Table 4.1 for three protocols: 1) the Write-Once protocol, 2) a protocol that includes modification 1 only, and 3) a protocol that includes modifications 1 and 4. Speedups for modifications 2 and 3 are nearly indistinguishable from the results for the protocols without these modifications, and are thus not shown. Results for each of the three levels of sharing considered in the GTPN study (1%, 5%, and 20%), are given for the first two protocols. For the third protocol, only the 5% sharing curve is drawn, since the other two curves are nearly identical (see Table 4.1(c).)

Below we discuss the results of our analysis, the accuracy of the mean value analysis estimates as compared with the GTPN results, and the relationship between the results in this study and results of independent evaluations of the protocols.

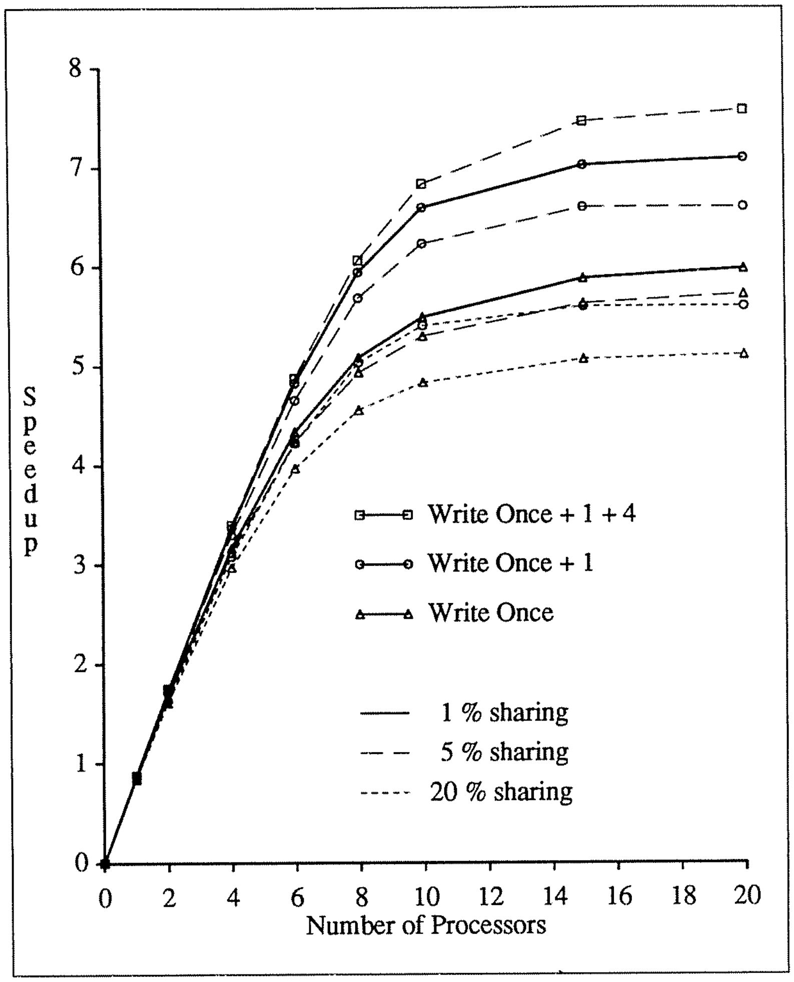

Figure 4.1: The Mean Value Analysis Performance Results

4.1. Model Results

As discussed in the next section, the mean-value analysis results are nearly identical to the results of the GTPN model. Furthermore, we are able to analyze the speedup for arbitrarily large systems using the MVA equations. (Solution of the GTPN model is impractical for more than ten or twelve processors.) Table 4.1c includes the MVA results for 100 processors, to verify that the performance does not change appreciably beyond twenty processors, for each curve shown in the figure. The asymptotic results indicate a greater potential gain for modification 4 than was evident from previous results for ten processors.

The conclusions we draw from figure 4.1 are as follows. Modification 1 is clearly advantageous for the workloads we have studied. Modification 4 is more advantageous as system size and the level of sharing increase, assuming this modification substantially increases the value of h_{sw}, as we have assumed in our input parameter values. Modifications 2 and 3 have little effect for the workload we investigated. We comment on this further in Section 4.4.

4.2. Agreement Between the Mean Value Analysis and GTPN Results

In this section we compare the performance estimates obtained from the mean-value equations with estimates obtained from the (expensive) solution of the detailed steady-state equations of the GTPN model.

Table 4.1(a) contains the numerical speedup estimates for the Write-Once protocol derived from the two models. Results for each of the three levels of sharing considered in the GTPN study are shown in the table. We find the speedup estimates of the mean value analysis are in excellent agreement with the speedup estimates of the GTPN for each sharing level. Nearly all MVA estimates are within 1% of the GTPN estimates, and the maximum relative error is 2.6%.

We also find very good agreement between the models (i.e., typically less than 5% relative error) for other performance measures, such as bus utilization and mean bus waiting time, which are not shown in the table. For example, in the 6-processor case, the GTPN and MVA estimates of bus utilization are approximately 81% and 77%, respectively. We note, however, that the approximate MVA equations generally underestimate bus utilization and overestimate memory and cache interference relative to the GTPN model.

Table 4.1(b) compares the speedup results from the MVA and GTPN models for the Write-Once protocol plus modification 1 of Section 2.2. Here again we find excellent agreement between the estimates, with most MVA results within 1% of the GTPN values, and a maximum relative error of 4.25%.

We investigated the accuracy of the MVA model further by validating it against the GTPN for each of the other three enhancements. In every case, the MVA model estimates agreed nearly exactly with the GTPN results. We thus conclude that, for the workload parameters in this set of experiments, the MVA model is as accurate as the more detailed GTPN model for assessing the relative merits of all of the proposed modifications to Write-Once. Table 4.1(c) further illustrates the point by giving the estimates from both models for the Write-Once protocol with modifications 1 and 4.

Table 4.1: Comparison Between MVA and GTPN Estimates

(a) Speedups for the Write Once Protocol

Sharing Level	Solution Method	Number of processors								
		1	2	4	6	8	10	15	20	100
1%	MVA	0.86	1.68	3.17	4.33	5.08	5.49	5.88	5.98	6.07
	GTPN	0.86	1.69	3.20	4.41	5.21	5.60			
5%	MVA	0.855	1.67	3.12	4.23	4.93	5.30	5.63	5.72	5.79
	GTPN	0.855	1.67	3.14	4.30	5.04	5.37			
20%	MVA	0.84	1.61	2.97	3.97	4.55	4.83	5.07	5.12	5.16
	GTPN	0.84	1.62	3.02	4.07	4.67	4.87			

(b) Speedups for Enhancement 1

Sharing Level	Solution Method	Number of processors								
		1	2	4	6	8	10	15	20	100
1%	MVA	0.875	1.73	3.37	4.82	5.94	6.59	7.02	7.09	7.04
	GTPN	0.875	1.73	3.37	4.84	6.00	6.72			
5%	MVA	0.87	1.71	3.30	4.65	5.68	6.23	6.59	6.64	6.60
	GTPN	0.86	1.71	3.31	4.71	5.76	6.31			
20%	MVA	0.85	1.63	3.08	4.22	5.03	5.40	5.63	5.66	5.62
	GTPN	0.85	1.65	3.15	4.39	5.19	5.58			

(c) Speedups for Enhancements 1 and 4

Sharing Level	Solution Method	Number of processors								
		1	2	4	6	8	10	15	20	100
1%	MVA	0.88	1.75	3.40	4.90	6.06	6.83	7.49	7.58	7.56
	GTPN	0.88	1.75	3.41	4.91	6.13	6.91			
5%	MVA	0.88	1.75	3.40	4.87	6.06	6.83	7.46	7.57	7.57
	GTPN	0.88	1.75	3.41	4.92	6.16	6.98			
20%	MVA	0.88	1.74	3.35	4.75	5.90	6.70	7.47	7.64	7.70
	GTPN	0.88	1.75	3.39	4.87	6.09	6.93			

These preliminary results indicate not only that the MVA model is quite accurate, but also that it is *as suitable as the GTPN for evaluating the potential performance gains of the various protocol modifications*. This result is surprising, given that the distinctions among the protocols are quite subtle. This is the first result known to the authors that indicates mean-value queueing analysis techniques may be applied to the evaluation of rather detailed architectural trade-offs. The computational efficiency of the MVA approach allows a wide range of design alternatives to be interactively investigated.

4.3. Accuracy of the Model Under Stress Tests

In the next set of experiments, we modified the workload parameters, in some cases assigning unrealistic values, in an attempt to find cases where the MVA equations are inaccurate. In particular, we experimented with cases that have a large amount of cache interference, since cache interference is represented much less precisely in the MVA model than in the GTPN.

In one experiment, we set the values of rep_p, rep_{sw}, and $amod_{sw}$ to 0.0, $csupply_{sro}$ and $csupply_{sw}$ to 1.0, p_{sw} to 0.2, and hit_{sw} to 0.1. The speedup estimates of the MVA model agreed, within 5% relative error, with the speedup estimates in the GTPN. This was the case in all of the experiments we performed in attempting to stress-test the MVA model. It appears that the MVA model is quite robust.

4.4. Agreement Between the Mean Value Model and Independent Evaluation Studies

It is generally difficult to compare results of the MVA and GTPN models with results of independent protocol evaluation studies, for two reasons. First, the parameter values used in experiments reported in the literature are not always fully specified. Second, if a different workload model is used, the mapping between parameters in the different workload models is generally not straightforward. In spite of these difficulties, we are able to compare our results with the results of independent studies in three cases.

The first comparison we are able to make is in the estimate of *processing power* for the protocol with modifications 1,2, and 3. Processing power is defined as the sum of the processor utilizations, over all processors in the system. Processing power can be computed from the MVA results by taking $\frac{\tau}{R} \times N$. Alternatively, processing power can be computed from the product of speedup and $\frac{\tau}{\tau + T_{supply}}$, which is $\frac{2.5}{3.5}$ or approximately 0.7143 for the workload assumptions in this paper. In either case, we compute a processing power of 4.32 for the protocol with modifications 1, 2, and 3, nine processors, 5% sharing, and parameter values equal to those in the appendix. The GTPN predicts a processing power of 4.1 for this case. Both results agree reasonably well with results of the simple analytical model in [PaPa84], for cache block size equal to four.

The second comparison we are able to make is in the estimate of relative bus utilization for Write-Once and a protocol with modifications 2 and 3. In this case, if the probability that a block is unmodified on a write hit decreases significantly in the protocol with modification 2, the MVA models predict a 10% increase in bus utilization for the Write-Once protocol, 99% sharing, and total loads which do not saturate the bus. This result agrees well with the trace-driven simulation results of Katz et. al. [KEWP85].

The final comparison we make is to the simulation results of Archibald and Baer's study [ArBa86]. An important discrepancy between the results of that study and the results in Figure 4.1 is in the performance estimates for modification 1 relative to modification 2. The simulation study shows a nearly equal performance benefit for each of these modifications. (For example, the Berkeley and Illinois protocols have nearly equal performance in most of their experiments.) Careful examination of their parameter values reveals that the value they use for $amod_p$ in most of their experiments is substantially higher than the value we assumed. If we set $amod_p$ to 0.95, as in many of their experiments, we also find the performance of modification 2 to be roughly equal to the performance of modification 1 for the 1% sharing case in Figure 4.1.

The agreement between the mean-value estimates and estimates from independent studies further increases our confidence in the accuracy of the MVA model.

5. Conclusion

Efficient, accurate tools for studying the performance implications of architectural design decisions are critically important.

In this paper we consider a family of dynamic cache consistency protocols for shared bus multiprocessor systems. The performance of these protocols has been studied extensively using detailed simulation models and Generalized Timed Petri Net models. These modeling techniques, while accurate, have running times measured in hours on 1 MIPS processors, for models of systems of only modest size.

We have devised a new modeling approach, based upon the specification and the iterative solution of sets of equations that express the mean values of interesting performance measures in terms of the mean values of certain model inputs. The equations are intuitive, in the sense that each can be explained simply in terms of the mechanics of the architecture being modeled. The solution technique is extremely efficient, requiring on the order of one second of CPU time for systems of arbitrary size. This makes it possible to explore a large design space quickly and interactively. The results are essentially as accurate as those of the previously existing techniques, which are dramatically more expensive.

We believe that we have convincingly demonstrated the surprising result that simple and efficient models can be used to study the performance of architectural alternatives that differ from one another in only quite subtle ways. The model can be put to good use for evaluating the protocols more thoroughly – all that is needed are workload measurement studies to aid in the assignment of parameter values.

The demonstration of the accuracy of our model is the principal thrust of our paper. Along the way, we have used the model to reproduce and extend the results in [VeHo86], in one case showing a greater benefit of protocol enhancement 4, which could not be anticipated from the existing results due to the inability to solve the more expensive models for large systems.

We believe that computer architects should seriously consider our "customized mean value equation" approach conducting future architectural performance studies. The approach is certainly applicable to the performance analysis of larger and more complex cache-coherent multiprocessors [Wils87, GoWo87]. It is most likely also applicable to realms other than cache consistency protocols.

We also wish to emphasize the utility of the more detailed GTPN and simulation tools for validating the results of the mean-value analysis for small systems. The authors are aware that there are cases where mean value analysis is inaccurate. Thus, validation against more detailed models is critically important when applying the technique to new problem domains.

References

[ArBa86] Archibald, J., and J.-L. Baer, ''An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors,'' *ACM Transactions on Computer Sysytems*, Vol. 4, No. 4, November 1986.

[DuBr82] Dubois, M., and F. A. Briggs, ''Effects of Cache Coherency in Multiprocessors'', *IEEE Trans. on Computers*, Vol. C-31, November 1982, pp. 1083-1099.

[Fran84] S.J. Frank, ''Tightly Coupled Multiprocessor System Speeds Memory Access Times,'' *Electronics*, Vol. 57, no. 1, January 1984, pp. 164-169.

[Good83] Goodman, J.R., ''Using Cache Memory to Reduce Processor-Memory Traffic,'' *Proc. of 10th Int. Symp. on Computer Architecture*, June 1983, pp. 124-131.

[GoWo87] Goodman, J.R., and P. Woest, ''The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor,'' to appear in *Proc. 15th Ann. Int'l. Symp. on Computer Architecture*, Honolulu, Hawaii, May 30 - June 2, 1988.

[GrMi87] Greenberg, A. G., and I. Mitrani, ''Analysis of Snooping Caches,'' to appear in *Proc. of Performance 87, 12th Int'l. Symp. on Computer Performance*, Brussels, December 1987.

[HoVe85] 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.

[KEWP85] 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.

[LZGS84] Lazowska, E. D., J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1984.

[MCCr84] McCreight, E., ''The DRAGON Computer System: An Early Overview,'' *NATO Advanced Study Institute on Microarchitecture of VLSI Computers*, Urbino, Italy, July 1984.

[PaPa84] 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.

[RuSe84] 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.

[Smit82] Smith, A.J., ''Cache Memories,'' *Computing Surveys*, Vol. 14, no. 3, pp. 473-530, September 1982.

[Smit85a] Smith, A. J., ''Cache Evaluation and the Impact of Workload Choice,'' *Proc. of 12th Int. Symp. on Computer Architecture,*, June 1985.

[Smit85b] 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.

[VeHo86] Vernon, M. K., and M. A. Holliday, ''Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets,'' *Proc. of Performance 86 and ACM SIGMETRICS 1986 Joint Conf. on Computer Performance Modeling, Measurement, and Evaluation*, Raleigh, N.C., May 1986, pp. 9-17.

[Wils87] Wilson, A. W., ''Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors,'' *Proc. 14th Annual Int'l. Symp. on Computer Architecture*, Pittsburgh, PA, June 2-5, 1987, pp. 244-252.

Appendix A

The following workload parameter values are used in the experiments in Section 4:

Parameter	Value		
τ		2.5	
$p_{private}$	0.99	0.95	0.80
p_{sro}	0.01	0.03	0.15
p_{sw}	0.00	0.02	0.05
$h_{private}$		0.95	
h_{sro}		0.95	
h_{sw}		0.5	
$r_{private}$		0.7	
r_{sw}		0.5	
$amod_{private}$		0.7	
$amod_{sw}$		0.3	
$csupply_{sro}$		0.95	
$csupply_{sw}$		0.5	
$wb_{csupply}$		0.3	
rep_p		0.2	
rep_{sw}		0.5	

Note that the value of rep_p is increased to 0.3 for Modification 1; rep_{sw} is increased to 0.6 for Modifications 2 or 3, and to 0.7 for a protocol with both modifications; and, finally, hit_{sw} is set to 0.95 for the protocol with modifications 1 and 4.

Appendix B

In this appendix we give the formulas for p, p', and $t_{interference}$, used in Section 3.1. We assume PSRWM, PSWHumod, SRMiss, SWMiss, SWHumod, SRMiss, SWMiss, and SWCSup, are defined as in [VeHo86]. The equations are as follows:

$$p = p_a + p_b,$$

where:

$$p_a = \frac{PSRWM}{PSRWM + PSWHumod} \times (SRMiss + SWMiss) \times 0.5$$

and

$$p_b = \frac{SWHumod}{PSRWM + PSWHumod} \times 0.5.$$

$$p' = p_b + p_a \times \frac{1}{\frac{n-1}{2}} \times (csupply_{sro} \times SRMiss + csupply_{sw} \times SWMiss)$$
$$\times [1-(rep_p \times p_{private} + rep_{sw} \times p_{sw})]$$

$$t_{interference} = 1.0 + \frac{p_a}{p_a + p_b} \left\{ \frac{1}{\frac{n-1}{2}} \times (csupply_{sro} \times SRMiss + csupply_{sw} \times SWMiss) \times [4.0 + (wb_{csupply} + SWCSup) \times 4.0] \right\}$$