ANALYTICAL MODELING OF DATA SHARING IN CACHE BASED MULTIPROCESSORS*

TECHNICAL REPORT NO. CENG 89-18

MICHEL DUBOIS AND JIN-CHIN WANG

JUNE 1989

*This work was supported by the National Science Foundation Grant DCCR-8709997

Analytical Modeling of Data Sharing in Cache-Based Multiprocessors

Michel Dubois and Jin-Chin Wang

Department of Electrical Engineering - Systems
University of Southern California
University Park, Los Angeles, CA 90089-0781
(213)743-8080

PART I: SHARED DATA CONTENTION IN A CACHE COHERENCE PROTOCOL

Abstract

In shared-memory multiprocessors, private caches are usually associated with each processor and coherence among caches is maintained in hardware by a cache coherence protocol on the memory bus. *Multithreading*, or the concurrent execution of the multiple processes forming a task is often supported in these systems. The efficiency of multiprocessor systems for a parallel algorithm depends to a large extent on the amount of sharing in the algorithm and on the effectiveness of the cache protocol for shared block accesses. Even if the cache sizes were infinite, the number of processors which can be connected to a bus would still be limited. In fact, the initial loading of data and instructions in each cache, as well as the active sharing of shared writable data among caches cause misses and bus traffic in the system with infinite caches.

In this paper, we analyze shared data contention in parallel algorithms and its effects on the performance of a cache coherence protocol under the assumption of infinite cache sizes. A simple program model for data sharing is introduced and an analytical closed-form solution is found for all components of the cache coherence overhead. We then study the overhead due to shared data contention in five parallel algorithms: the iterative Jacobi algorithm, the iterative S.O.R. algorithm, the parallel quicksort, and the shuffling and non-shuffling FFTs. Finally, these overheads are compared with the predictions of the analytical model.

1. INTRODUCTION

In modern multiprocessors, a given algorithm may be decomposed into cooperating processes that run in parallel [3]; this technique is called *multitasking* or *multithreading*. In a shared-memory multiprocessor processes working in parallel on the same algorithm cooperate through the sharing of data in memory. Usually, in a shared-memory multiprocessor a cache [19] is associated with each processor, in order to reduce both memory access latency and memory-bus traffic. Shared data may be cached provided a hardware protocol maintains consistency among multiple copies of the same data in different caches. By caching shared data one hopes to improve the hit ratio, and therefore the execution speed of the multiprocessor.

Three techniques are possible to analyze the performance of cache-based multiprocessors: measurements on an existing system, trace-driven simulations, and simulations or analytical models based on a program behavior model. While measurements are interesting because the results bear on *real* programs and *real* machines, they have definite shortcomings. Mostly, measurements are specific to the multiprocessor system on which they are taken. Trace-driven simulations are very compute-intensive, and therefore an exhaustive simulation study for all possible values of the parameters is not really feasible. In [17], trace-driven simulation results are presented, but the problem under investigation is such that the activity of only one processor has to be traced. Recently, the results of the trace-driven simulations of a few parallel programs have appeared in the literature [12,14,1]

Early work on the analytical evaluation of cache-based systems was done by Patel [18] and Briggs and Dubois [5]; these papers either ignored the cache coherence effect or assumed that shared data are not cached. In order to include the effect of data coherence in the models, several authors have taken the approach of modeling the workload with an analytical program model [20]; the workload model is then used in a simulation or in an analytical model. In [10], Dubois and Briggs introduced a model for multiprocessor program behavior and derived a closed-form solution in the case of a multiprocessor system with finite caches and LRU (Least Recently Used) replacement policy. Archibald and Baer [4] published simulation results comparing various cache protocols; the program model is similar but more complex, and no closed-form solution is proposed. In [21] Vernon and Holliday introduced a timed Petri net model driven by the same program model as Dubois and Briggs's; no closed-form solution was proposed.

One aspect that has been neglected is the correlation between the predictions of the program models and actual parallel algorithms. The basic contribution of our paper is to introduce and justify a new program model for data sharing and to compare analytical model predictions to simulation results for actual algorithms, in the case of one cache coherence protocol and under some assumptions. Such a correlation is critical to the credibility of program model predictions.

The approach taken in this paper is to analyze a *baseline system, the infinite cache*

model. This model is much simpler to study than the *real* system with finite caches, mostly because it is described by fewer parameters. Because of this simplicity, a closed-form solution can be found for analytical program models, shared data contention can be more easily analyzed for specific algorithms, and the amount of output data from the performance models is manageable. Deviations from the baseline model caused by features such as cache size, organization or replacement policies can be studied in isolation, through simulations. The infinite cache model was also adopted in the simulation studies [14,1].

2. INFINITE CACHE MODEL AND GENERAL ASSUMPTIONS

Several simplifying assumptions are made in this paper. We first list them, then we discuss their validity.

Assumption 1: The size of all caches is infinite.
Assumption 2: The models are in steady-state. Initial transients are not included.
Assumption 3: Process preemption and migration are disallowed; i.e., a process executes from start to finish on the same processor and without interruption.

The major motivation for studying the infinite cache model is its simplicity. Most parameters of the cache do not affect the model prediction, such as cache size, cache organization or cache replacement policy; the resulting models are therefore parsimonious.

There is another reason why the infinite cache model is a compelling model. Present trends in memory chip sizes indicate that fast and large caches are becoming possible. In these caches, most of the misses are due to the initial loading of the data, and to coherence invalidations. It is expected that the infinite cache model will become more and more relevant as the level of integration of memory chips increases.

Modeling transient effects in an infinite cache is not difficult, but the models are not very interesting. For example, at program start, caches are empty; every block referenced in a parallel algorithm must first be brought into one of the caches; this initial miss is not counted in the models. The number of these initial misses is simply equal to the total number of different blocks accessed during the whole execution of the parallel algorithm.

The third assumption may be the most restrictive. We will reconsider this assumption in Section 8.

3. CACHE COHERENCE PROTOCOL

The cache coherence protocol considered in this paper is an invalidation based protocol described for example in Hwang and Briggs's book [15, pages 521–525]; other protocols have been designed [4], and the techniques described in this paper can be applied to any one of them [22].

Multiple copies of the same cache block may be present in different caches, provided the copies are Read-Only (RO copies), that is, provided no processor has modified any word in

the block. If a processor needs to modify a word in a block, it must obtain a Read-Write copy (RW copy), i.e. a unique copy of the block: this may involve invalidating copies of the block in other caches. Usually, a block containing only instructions, private data or shared constants will be tagged as RO, while blocks containing shared writable data may be tagged as RW. Therefore, we distinguish between S-blocks and P-blocks. An S-block contains data items accessed and modified by different processors while a P-block contains data items accessed by only one processor or Read only data. In the protocol selected for this study, the following cache events on an S-block may occur in a multiprocessor systems with infinite caches and in steady-state (refer to Figure 1):

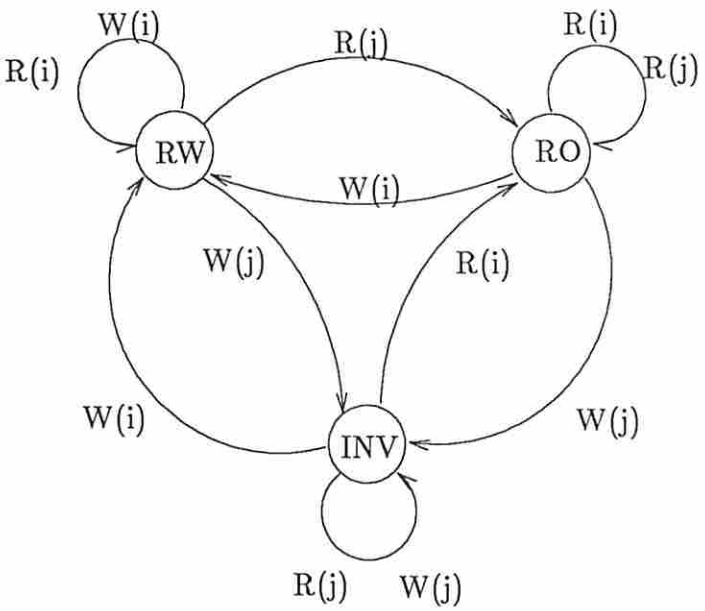


Figure 1: State diagram for a given block in cache i (infinite cache assumption)

R(i) : Read block by processor i.
R(j) : Read block by processor j.
W(i) : Write to block by processor i.
W(j) : Write to block by processor j.

1. *Miss:* this event occurs when the data is referenced and is not present in the cache. We denote this event as M (Miss). All misses occurring as a result of the following events are accounted for as M events. Blocks are always loaded from the main memory on a miss.

2. *Transition from RO to RW:* this event occurs when a processor needs to modify a block already present in another cache as RO; a miss may occur and an invalidation must be sent to the processor(s) possessing the RO copy(ies); we denote this event as IN_RO (for INvalidation of RO copy(ies));

3. *Transition from RW to RO:* this event occurs when a processor reads a block present in another cache as RW; besides the occurrence of a miss, a signal must be sent to the cache possessing the RW copy and this cache must write the block back to main memory; we denote this event as CS_RW (Change State of a RW copy).

4. *Transition from RW to RW* in a different cache: this event occurs when a cache needs to modify a block which is owned as RW by another cache; it implies a write back to main memory, a miss and an invalidation; we denote this event as IN_RW (INvalidation of a RW copy).

When writable blocks are actively shared, copies must be transferred among caches and invalidation signals must be sent. As the number of processors actively sharing a block increases, the invalidation activity usually increases.

4. ANALYTICAL MODELS

The access pattern to shared data in multiprocessor systems depends on the algorithm. We can distinguish between two broad classes of shared variables: synchronization data (such as locks) and other shared operands. Synchronization data are used to coordinate process execution or to protect shared operand accesses.

Kung [16] classifies multitasked algorithms into *synchronized* and *asynchronous* algorithms. In *asynchronous* algorithms, accesses to shared operands are not protected and each processor may access the data as it needs them. In *synchronized* algorithms, accesses to shared data are restricted, either by explicit synchronization or simply by structuring the forking and joining of processes. In *synchronized* algorithms, shared writable data are accessed either in **critical sections** [3] (only one process can access the data at a time either to Read or to Write) or in **semi-critical sections** [6] (multiple processors can read a data item at a time, but if a process has to modify the data item, it must do so in mutual exclusion). Figure 2 (a) and (b) illustrates both access patterns. In this Figure, only accesses to a specific shared datum are shown.

4.1 Analytical Model for one S-block

The program model is derived from the model in [20]. We had to extend this model because it did not capture the locality of references to shared writable data. In another paper [8], we presented two additional program models for which the effect of cache coherence can be solved analytically and which take into account the accesses made in critical

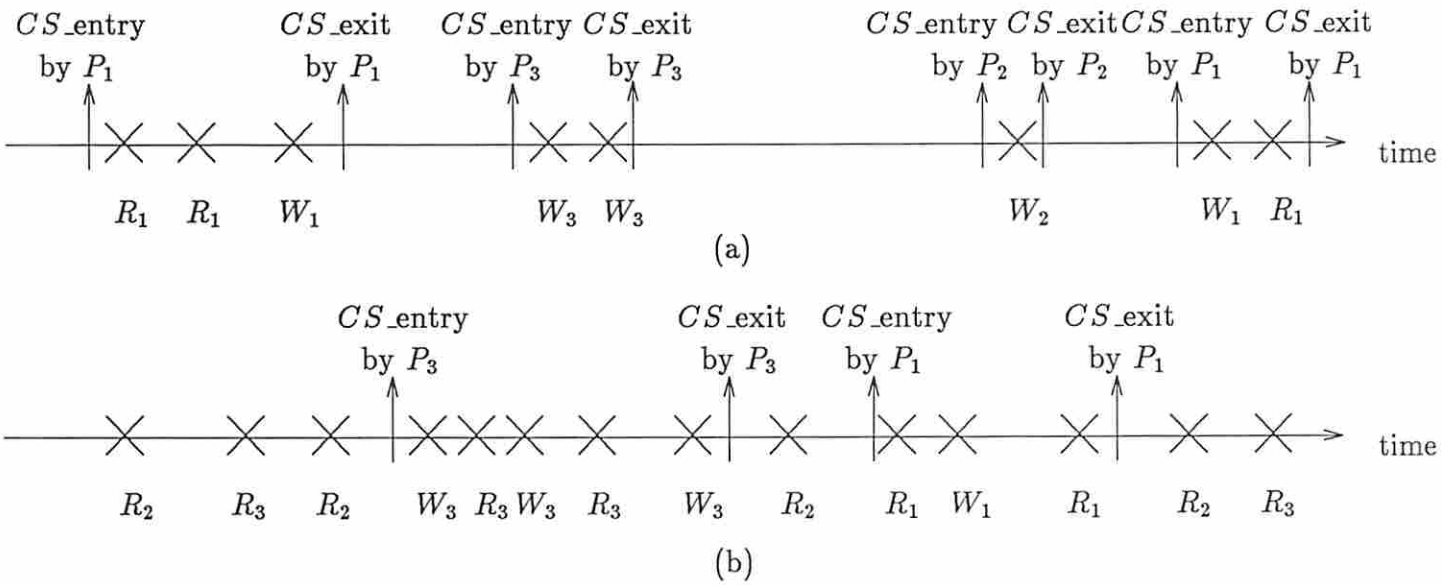


Figure 2: (a) Access pattern to a shared writable datum protected by critical sections. (b) Access pattern to a shared writable datum protected by semi-critical sections.

R_j: Read access to shared datum X by processor j
W_j: Write access to shared datum X by processor j

sections and semi-critical sections. All these program models can be defined as a special case of the following model. The program model that we are about to define assumes that accesses by one processor to a shared writable block are done in uninterrupted bursts. Besides modeling critical section accesses, the burst model takes into account the locality of reference on S-blocks. We use the same notation as in [10].

The P processors execute independent streams of instructions and generate homogeneous streams of references. S-blocks belong to different *sets*; all S-blocks in a set are accessed with the same pattern, even if they are accessed by different processors; the program model, model parameters and coherence overheads are identical for all the S-blocks in a set.

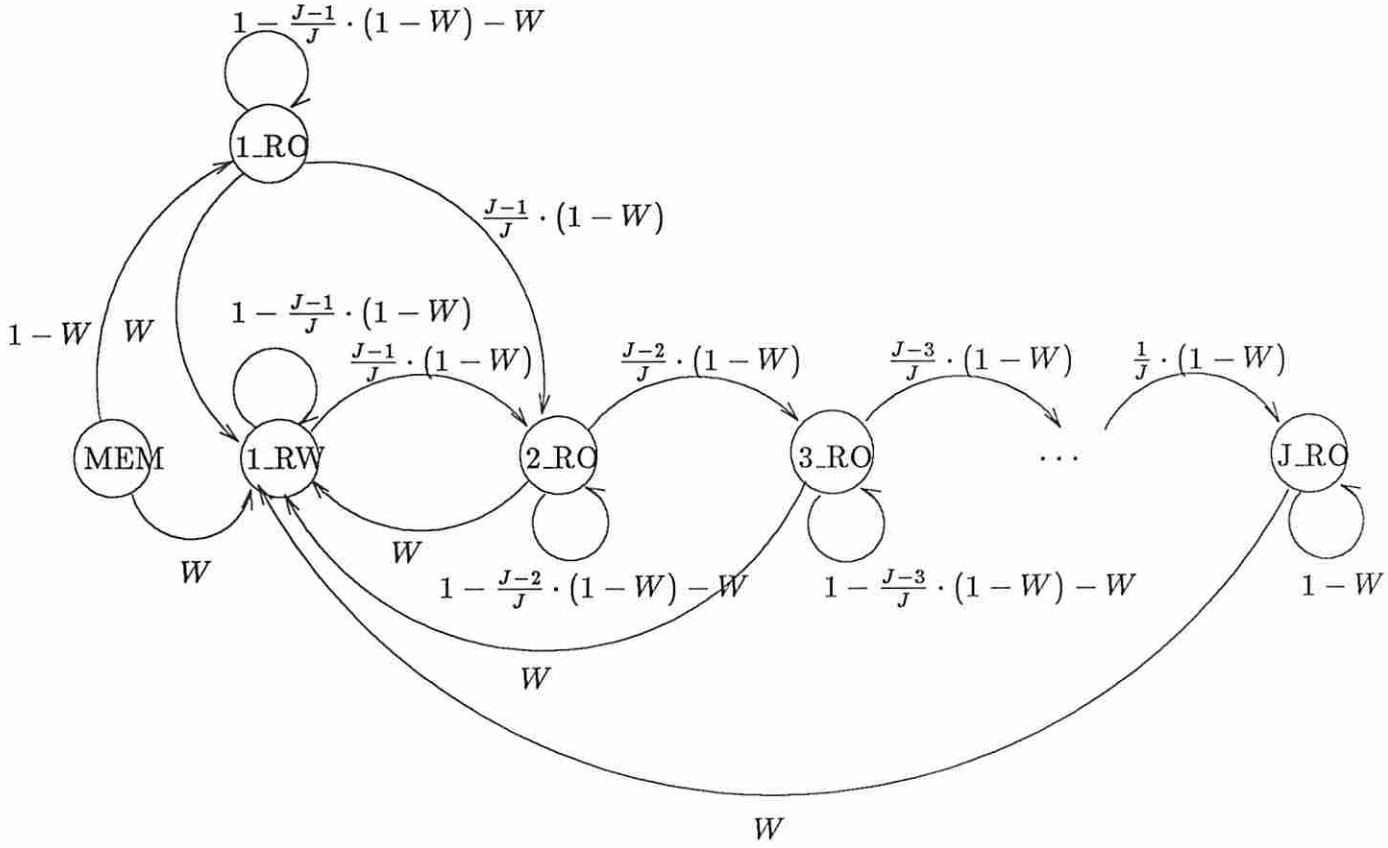
Let q_s be the fraction of references to S-blocks. The fraction of S-block accesses that are for a particular S-block i is p_i with i=1,...,N_s and N_s is the total number of shared writable blocks. S-block i is shared by J_i processors ($J_i \leq P$, the total number of processors in the system). Processors access an S-block i in bursts. l_i is the average burst size, i.e. the average number of accesses to the block during an access burst. An isolated access is counted as a burst of size one. The average burst size can be found by dividing the total number of references by the number of access bursts. For example, for the program fragments of Figure 2 the average burst sizes are $l_i = 2$ (Figure 2(a)) and $l_i = 1.75$ (Figure 2(b)), assuming that one cache block contains only one data element.

The fraction of processor references which start an access burst for a given S-block i is $\frac{q_s \cdot p_i}{l_i}$. The basic approximation of the analytical model is that access bursts are independent from one another. When a processor completes an access burst, all the J_i processors have the same probability of starting the next access burst to the shared block. We designate by W_i the probability that the block is modified during an access burst. Because of the infinite cache assumption, there is no interference among cache accesses to different blocks and the events occurring for one block are independent of the events occurring for any other block; the state transitions of S-block i can be observed in isolation.

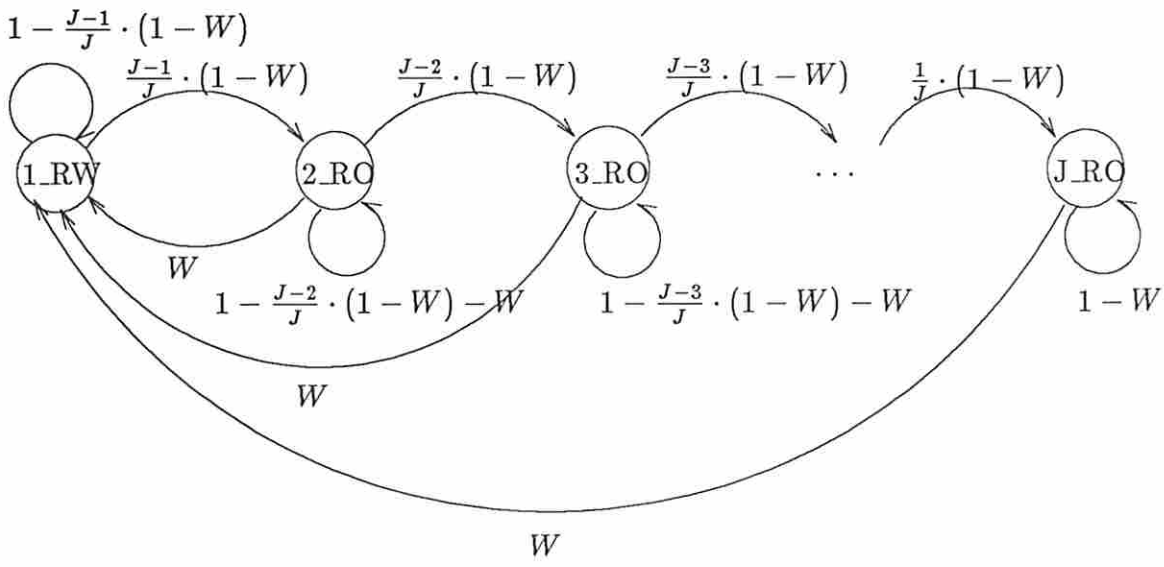
The *global state* of an S-block i is described by the number of caches possessing a copy of the block and by the status RO or RW of the block. The global states are denoted by 1_RW, 1_RO, 2_RO,..., J_i_RO. We can ignore the identity of specific processors because the multiprocessor is homogeneous and symmetric.

The Markov chain for the state transitions of S-block i is shown in Figure 3 (we have dropped the index i in the Figure for clarity. Note that all parameters are for a given S-block i). The state of the Markov chain is the global state of the block *whenever an access burst is completed* (except for the state MEM, which is the state of the block before the first reference to it). It is clear from the Figure 3(a) that states MEM and 1_RO are transient states. Figure 3(b) shows the reduced Markov chain where the transient states have been removed. We will only solve the Markov chain of Figure 3(b). A state transition occurs in this state diagram every time a burst of accesses is completed by one processor. The transition probabilities from state k_RO, $k < J_i$, are found as follows.

1. From state k_RO to state $k+1\_RO$: The probability of this transition is the product



(a)



(b)

Figure 3: (a)Markov chain for the state transitions of an S-block shared by J processors (including transient states). (b)Markov chain for the state transitions of an S-block shared by J processors (without transient states).

of the probability that the next burst contains only Read accesses, $(1 - W_i)$, and of the probability that the access burst is made in one of the $J_i - k$ other caches, $(\frac{J_i - k}{J_i})$.

2. From state k_RO to state k_RO: This is the case when the next access burst contains only Read accesses in one of the k caches. The transition probability is $(1 - W_i)\frac{k}{J_i}$.

3. From state k_RO to state 1_RW: This is the case when the next access burst modifies the block. The transition probability is W_i.

The transition probabilities from states 1_RW and J_i_RO are derived from similar arguments.

This finite state Markov chain is aperiodic and irreducible. Let's denote by $Pr(1)$ and by $Pr(k)$, k=2,...,J_i, the state probabilities of state 1_RW and states k_RO respectively. The state probability distribution is given by the set of equations: (see for example [2])

$$Pr(k) = \frac{(J_i - k + 1)(1 - W_i)}{(J_i - k)(1 - W_i) + J_i W_i} Pr(k-1), \quad for\ k = 2, ..., J_i \tag{1}$$

and

$$Pr(1) = \frac{J_i W_i}{(J_i - 1)(1 - W_i) + J_i W_i} \tag{2}$$

With these state probabilities, one can compute the probability of occurrence of each coherence event. When there are k copies in k processor caches a miss occurs at the beginning of a new access burst, i.e. at a state transition in Figure 3(b), if the next processor to start an access burst is one of the $(J_i - k)$ processors without a copy in their cache. Therefore, the fraction of references to S-block i which miss in the cache is equal to the fraction of state transitions causing a miss divided by the average burst length l_i.

$$M_i = \frac{1}{l_i}\left[Pr(1) \cdot \frac{(J_i - 1)}{J_i} + Pr(2) \cdot \frac{(J_i - 2)}{J_i} + ... + Pr(J_i - 1) \cdot \frac{1}{J_i}\right] \tag{3}$$

After some transformations, one finds simply (see Appendix A):

$$M_i = \frac{1}{l_i} \frac{(J_i - 1)W_i}{1 + (J_i - 1)W_i} \tag{4}$$

In the Markov graph of Figure 3(b) a transition from state 1_RW to state 1_RW results from three possible sequences of events:

1. the processor owning the RW copy of the block has started a new burst of accesses for the same block; no event is recorded for S-block i in this case;

2. a different processor has started an access burst for S-block i and its first access to the block is a Write; an event of type IN_RW must be recorded for S-block i;

3. a different processor has started an access burst for S-block i and its first access to the block is a Read followed by a Write; one event of type CS_RW followed by one event of type IN_RO must be recorded.

In order to differentiate between the 2nd and the 3rd cases, we have to introduce a new factor f_i, which is the fraction of Write bursts[1] such that the first access is a Write. f_i can easily be computed from a string of references. For example, in Figure 2(a), $f_i = .75$, and, in Figure 2(b), $f_i = .5$. Taking into account this problem, one can derive the fraction of references to S-block i that result in a given event.

An invalidation of RO copies occurs whenever an access burst modifies a block in an RO state. It also occurs in a transition from 1_RW to 1_RW, provided the second access burst is executed by a different processor and starts with a Read. Therefore, the fraction of accesses to S-block i invalidating RO copies in other caches is given by:

$$I_i = \frac{1}{l_i}\left[W_i(1 - Pr(1)) + W_i(1 - f_i)Pr(1)\frac{J_i - 1}{J_i}\right]$$

A change of state from RW to RO occurs whenever a burst leaving the block in state 1_RW is followed by a burst starting with a Read access by a different processor. Therefore, the fraction of references to S-block i changing the state from RW to RO is:

$$C_i = \frac{1}{l_i}\left[Pr(1)(1 - W_i)\frac{J_i - 1}{J_i} + Pr(1)W_i(1 - f_i)\frac{J_i - 1}{J_i}\right]$$

Finally, an invalidation of a RW copy occurs whenever an access burst leaving the block in state 1_RW is followed by a Write from any other processor. The fraction of references to S-block i causing such an event is therefore:

$$D_i = \frac{1}{l_i}Pr(1)f_iW_i\frac{J_i - 1}{J_i}$$

In these equations, $Pr(1)$ is given by equation (2).

4.2 System Effects

We use the results of the previous section to model the effect of cache coherence on the overall system performance under the assumptions of Section 2. Two performance measures are derived: the miss ratio and the average coherence penalty.

4.2.1 Miss ratio

In the infinite cache model, if we neglect the transients, the miss rate is given by the miss rate on shared writable blocks, that is,

$$M = q_s \sum_{i=1}^{N_s} p_i M_i \tag{5}$$

[1] By definition, a Write burst is an access burst containing *at least* one Write access.

where N_s is the total number of shared writable blocks. The value for M_i is obtained by applying equation (4) and depends on different values of the parameters for different S-block i. In many cases, the terms in the above sum can be clustered by grouping the shared writable blocks into sets; within a set all blocks are referenced with the same pattern, and therefore have the same value of M_i.

To find M from equation (5), one need to specify the model parameters for all sets of blocks. In the studies presented in [10,4,21], there is only one set of parameters. Implicitly, it is assumed that the models can be applied to a single *average* set including all the shared writable blocks. Parameters are therefore computed as averages. For the five examples presented in Sections 5 and 6, this approach is shown to be acceptable.

4.2.2 Average Coherence Penalty

A processor runs at maximum speed when no cache misses or coherence events occur. To each coherence event corresponds an average penalty, λ_{EVENT}. The penalty associated with an event is defined as the average time that a processor is blocked at the occurrence of the event. The average coherence penalty per memory reference to S-block i is:

$$\lambda_i = M_i\lambda_M + I_i\lambda_{IN\_RO} + C_i\lambda_{CS\_RW} + D_i\lambda_{IN\_RW}$$

M_i, I_i, C_i, and D_i were defined in Section 4.1.

If we neglect the transients, the average coherence penalty in the infinite cache system is given by the sum of the coherence penalties on each shared writable block i:

$$\lambda_{total} = q_s \sum_{i=1}^{N_s} p_i\lambda_i \tag{6}$$

As for the system miss rate, S-blocks can be clustered into a few sets in which blocks have the same average coherence penalty. The average penalty could also be approximated by the penalty for an *average* block.

The average coherence penalty adversely affects the processor efficiency. In powerful and expensive main frame multiprocessors, any loss of processor efficiency is critical for the performance/cost ratio of the system.[2]

5. APPLICATION TO MULTITASKED ALGORITHMS (CACHE BLOCK SIZE IS ONE)

If the cache block size is one data element, then the values of the parameters are straight forward. This section deals with a cache block size of one. In Section 6, we will investigate the block size effect.

[2] If T_1 is the mean execution time of an instruction in the uniprocessor system (in microsecond), and if r is the average number of memory references per executed instruction, the average instruction execution time is $T_1 + r\,\lambda_{total}$, and the MIPS rate (Million of Instructions Per Second) per processor is $MIPS\ rate = \frac{1}{T_1 + r\lambda_{total}}$

.

We compare the model predictions with the simulation of specific algorithms running on shared-memory multiprocessors in which each processor has a private data cache of infinite size. To simulate the parallel algorithms a simulation methodology described in [11] was applied. In this methodology, the algorithm is actually executed on a uniprocessor and the multiprocessing effect is obtained by executing the process of each simulated processor in turn. The simulator *switches* from one simulated processor to the next on each data access and synchronization primitive execution. Many simulation results can be derived analytically, because of the simplicity. These analytical derivations whenever they are possible are presented in Appendix B.

The class of algorithms selected is a class of iterative algorithms which has been studied in many previous papers under similar assumptions (for example, see [7]). These algorithms exhibit shared data contention and can be mapped easily to the analytical models of the previous section.

5.1 Relaxation Algorithms for Partial Differential Equations

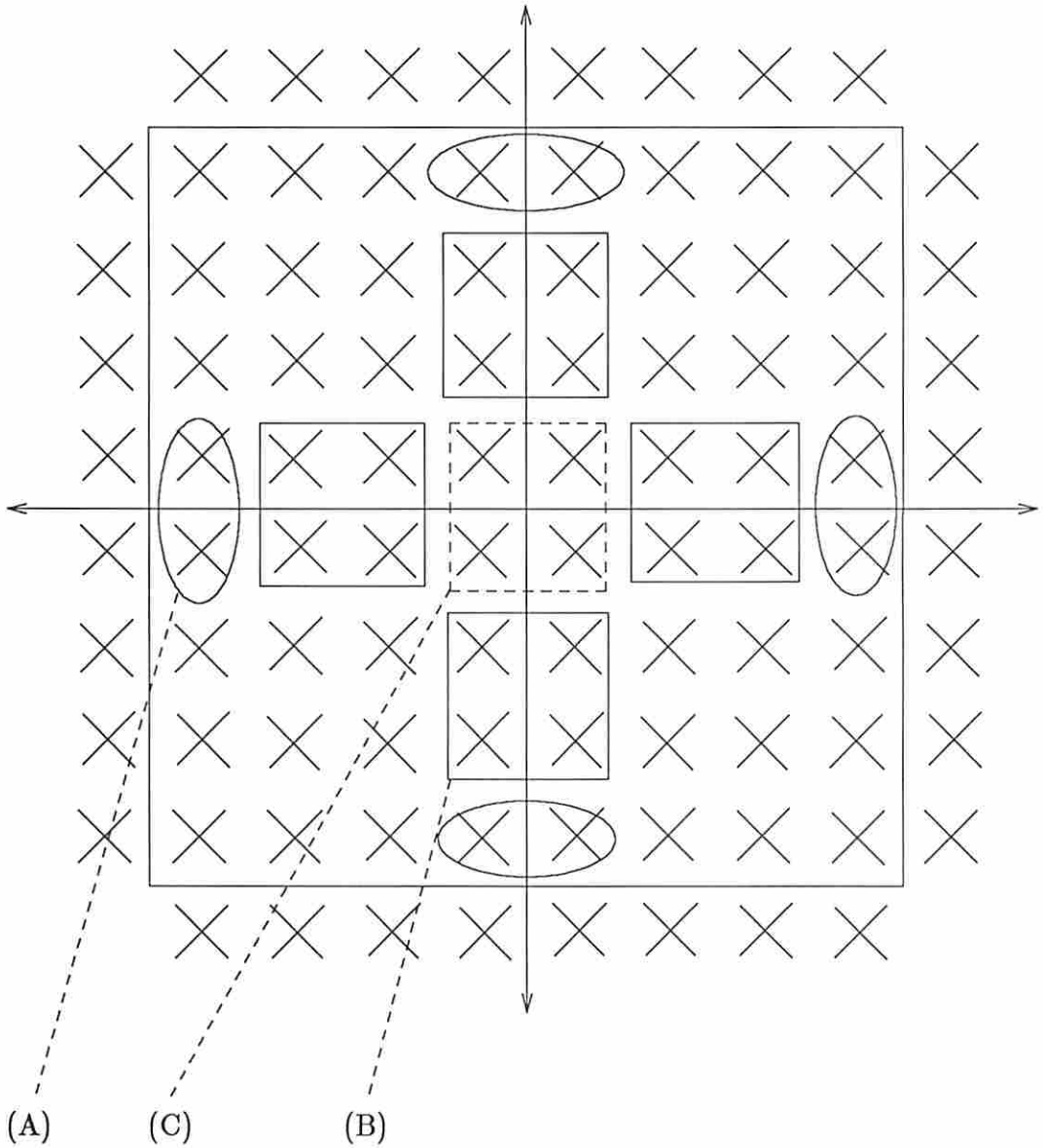
We consider two iterative schemes [23], the Jacobi and the Successive Over Relaxation (S.O.R.) algorithms. In the Jacobi iteration, the computation consists in repetitively updating each point of a grid as follows:

$$x_{i,j}{}^{(K+1)} = \frac{1}{4}\left[x_{i+1,j}{}^{(K)} + x_{i-1,j}{}^{(K)} + x_{i,j+1}{}^{(K)} + x_{i,j-1}{}^{(K)}\right].$$

The Jacobi iteration requires to maintain two grids. In each iteration, each point of one grid is updated by using the values of the 4 neighbors in the other grid. Then the processors synchronize and the two grids are interchanged. For a M $\times$ N grid, there are 2(M+N) boundary grid points and these points are not modified during execution; these points are Read-Only and can be treated as P-blocks. The grid points adjacent to these boundary points are called *outer* grid points. The reference pattern to outer grid points is different from the pattern to *inner* grid points. Consider, for example, two square grids of size 8 $\times$ 8 for which we allocate one processor to each subgrid of size 4 $\times$ 4, as displayed in Figure 4. The sets of shared writable data are circled in Figure 4. There are only three sets of S-blocks (in the sense of Section 4.1), (1) inner grid points with $J = 2$, (2) outer grid points with $J = 2$, and (3) inner grid points with $J = 3$.

Shared writable data are accessed in semi-critical section in the Jacobi iteration algorithm. In one iteration, the shared data in one grid are Read-Only and are accessed by different processors, and in the next iteration they are modified by a single processor in a critical section phase. Since the models developed in this paper are for infinite caches in steady-state, we consider iterations n and n+1 where n>1 and we assume that the data caches are large enough to contain the two subgrids accessed by each processor.

The parameters of the model are easily derived. Consider, for example, a grid size of 128 $\times$128 and $P = 4$. Table 1 shows the computed value of the model parameters and the



(A) outer points, J=2
(B) inner points, J=2
(C) inner points, J=3

Figure 4: The three data sets in the Jacobi iteration.

model predictions for the overall miss ratio and average penalty associated with each of the 3 sets. The exact values have also been computed and verified by simulation. Analytical derivation of these values and of the model parameters are explained in Appendix B.

In Tables 1 through 5, the column labeled p_s contains the fraction of processor references to the S-blocks in the set; i.e., $p_s = q_s \sum_{set} p_i$; the column labeled n_s contains the total number of S-blocks in the set; the *miss* and *penalty* columns contain the total contribution of the data set to the overall data miss ratio and to the average coherence penalty, that is, miss=$p_s \cdot M_i$ and penalty=$p_s \cdot \lambda_i$, where i refers to any S-block in the set. The unit for the penalties is the average penalty for a miss. We have chosen the following penalties for each event: $\lambda_M = \lambda_{CS\_RW} = \lambda_{IN\_RW} = 1$, $\lambda_{IN\_RO} = 0.5$ (remember that events CS_RW and IN_RW cause one miss and one write-back access to occur). From Table 1, it appears that the inner grid points shared by two processors dominate the overall miss ratio and average coherence penalty. Table 1 also shows good agreement between model results and simulation values.

Table 1: Jacobi Iterative Algorithm (P=4, 128 $\times$ 128 grid)

Set	J	W	l	f	p_s	n_s		miss	penalty
Inner	2	0.200	1	1	0.03027	992	model	0.00505	0.01211
							simulation	0.00605	0.01261
Outer	2	0.250	1	1	0.00039	16	model	0.00008	0.00019
							simulation	0.00010	0.00020
Inner	3	0.200	1	1	0.00024	8	model	0.00007	0.00013
							simulation	0.00010	0.00014

In the S.O.R. algorithm, only one copy of the grid is needed and iterates are updated according to the red/black ordering: grid elements which are in even positions (the sum of the indexes is even) are tagged as black, others as red. Each iteration proceeds in two sweeps. The red elements are updated in the first sweep, and the black elements are updated in the second sweep. After each sweep, each processor has to synchronize with at most 4 neighbors. Each processor has the same number of red and of black iterates. The equation for the update of an iterate in the (K+1)th iteration is

$$x_{i,j}^{(K+1)} = (1 - \omega)\, x_{i,j}^{(K)} + \frac{1}{4}\,\omega\, \left[x_{i+1,j}^{(K)} + x_{i-1,j}^{(K)} + x_{i,j+1}^{(K)} + x_{i,j-1}^{(K)}\right],$$

where ω is the relaxation factor.

During one sweep of the algorithm, some shared grid points are read by multiple processors, and some others are read and modified by one processor. As for the Jacobi iteration, there are 3 sets of shared writable grid points. Table 2 displays the comparison between the model and the simulation of the S.O.R. iteration (see Appendix B for the derivations

of the model parameters and of the actual miss rate and coherence penalty). Again, it appears that the set containing the inner grid points shared by two processors dominates the miss and penalty results. There is close agreement between the model predictions and the exact values.

Table 2: S.O.R. (P=4, 128X128 grid)

Set	J	W	l	f	p_s	n_s		miss	penalty
Inner	2	1.20	2	0	0.03027	496	model	0.00420	0.01009
							simulation	0.00505	0.01261
Outer	2	1.25	2	0	0.00041	8	model	0.00007	0.00015
							simulation	0.00008	0.00021
Inner	3	1.20	2	0	0.00024	4	model	0.00006	0.00011
							simulation	0.00008	0.00014

5.2 Quicksort

Quicksort is a *divide-and-conquer* algorithm, which sorts a file A[1], A[2], ..., A[N] by rearranging it to make the condition that A[1], ..., A[j-1] $\leq$ A[j] $\leq$ A[j+1], ..., A[N] hold for some j, and by recursively applying the same procedure to the subfiles A[1], ..., A[j-1] and A[j+1], ..., A[N]. In a multiprocessor, at the end of each splitting phase, the larger subfile is processed by the same processor and a descriptor of the smaller subfile is sent to a global job queue. An idle processor keeps on checking the global job queue and grabs a subfile descriptor when the global job queue is not empty.

In this algorithm, while a processor splits a subfile, no other processor accesses any data item in the subfile. Therefore shared data are always accessed in critical sections. Because the behavior of quicksort is data dependent, every data item in the file can be accessed by all processors. Table 3 shows the results for one 32K random file and P=8. Seven sets of S-blocks were identified based upon the number of processors accessing them. The parameters for each set are average values. Because of the sheer complexity, we cannot analytically derive the values of parameters for dynamic quicksort. We have added an additional procedure in our simulator to extract the parameter values.

5.3 Fast Fourier Transform (FFT)

The one-dimensional non-shuffling FFT algorithm for N data items is represented by a butterfly graph with $log_2 N$ stages. A bit-reversal permutation is applied at some point of the algorithm, so that the results are stored in the same order as the initial data items. Let $s(k)$, k=0,1,2,...,N-1 be N samples of a time function. The DFT (Discrete Fourier

Table 3: Dynamic Quicksort Algorithm (P=8, file size=32K)

Set	P	J	W	l	f	p_s	n_s		miss	penalty
Set2	8	2	0.3696	10.361	0	0.0147	402	model	0.00038	0.00096
								simulation	0.00055	0.00136
Set3	8	3	0.5088	7.2986	0	0.1279	3798	model	0.00884	0.01950
								simulation	0.00978	0.02345
Set4	8	4	0.5624	5.6840	0	0.3258	10360	model	0.03599	0.07671
								simulation	0.03733	0.08672
Set5	8	5	0.5872	4.6510	0	0.3480	11648	model	0.05248	0.10995
								simulation	0.05369	0.12105
Set6	8	6	0.5989	3.8610	0	0.1581	5609	model	0.03070	0.06356
								simulation	0.03135	0.06889
Set7	8	7	0.5927	3.3003	0	0.0244	915	model	0.00576	0.01174
								simulation	0.00593	0.01261
Set8	8	8	0.5984	3.0241	0	0.0008	30	model	0.00021	0.00042
								simulation	0.00022	0.00045

Transform) of $s(k)$ is defined to be the discrete function $x(j)$, j=0,1,2,...,N-1, where

$$x(j) = \sum_{k=0}^{N-1} s(k)\ e^{\frac{2\ \pi\ i\ j\ k}{N}}$$

where j=0,1, ..., N-1 and $i = \sqrt{-1}$.

In the non-shuffling FFT algorithm, we divide the array of N items into P chunks containing $\frac{N}{P}$ consecutive items. Each processor computes the FFT for its chunk, containing $\frac{N}{P}$ data items. For N=16 and P=4, the non-shuffling FFT algorithm is illustrated in Figure 5. In general, each S-block is shared by $log_2 P + 1$ processors and the algorithm can be divided into two parts. In the first part, i.e., in the first $log_2 \frac{N}{P}$ stages of the butterfly, every shared block is accessed by one processor. In the second part, i.e., in each of the last $log_2 P$ stages of the butterfly, each shared block is first read by two processors and then modified by a single processor in a critical section. Since there is no coherence activity in the first part of the non-shuffling FFT algorithm, we examine the second part of the algorithm. Synchronization is necessary in this algorithm and is denoted by dotted lines in Figure 5. In general, 2 $log_2 P$ synchronization points are needed in the second part (If the algorithm used two copies of the array and alternated between the copies only $log_2 P$ synchronization points would be needed.)

There is only one set of shared writable blocks in this algorithm. Each S-block is shared by $log_2 P + 1$ processors, that is, $J = log_2 P + 1$, and l, the number of accesses in a burst, is one. W is equal to $\frac{1}{3}$. Table 4 shows the prediction of the model for the case where

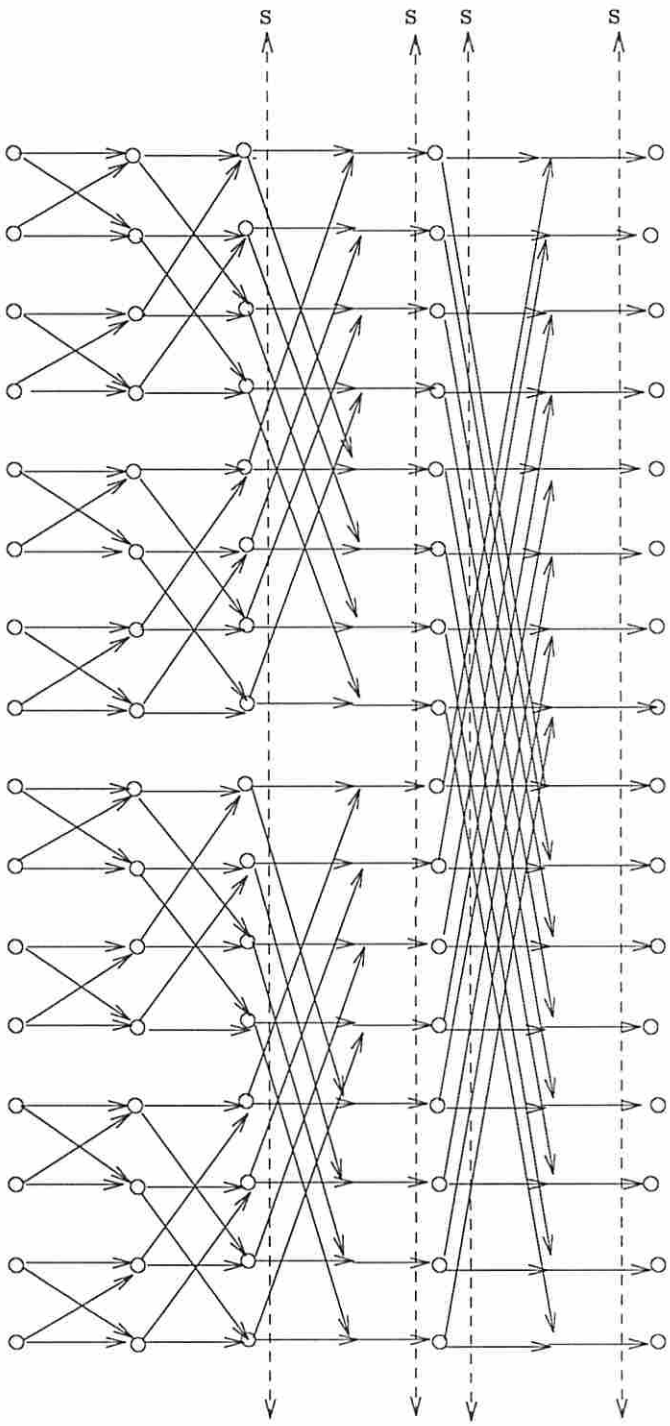


Figure 5: Non-shuffling FFT algorithm for P=4 and N=16.

P=4 and 16 (i.e., J=3 and 5) and the data set size is 64K. The simulated values show good agreement with the model for P=4, but the precision of the model deteriorates for P=16. Actually, simulations show that the miss rate and average penalty are independent of the number of processors, while the model predicts otherwise. Shared data contention is very high in the second part of the non-shuffling FFT: the global hit ratio for data is only around 60% because of invalidations.

Table 4: Non-shuffling FFT Algorithm (P=4,16, and 64K data elements)

P	J	W	l	f	p_s	n_s		miss	penalty
4	3	0.333	1	1	1.000	65536	model	0.4000	0.7810
							simulation	0.3333	0.8333
16	5	0.333	1	1	1.000	65536	model	0.5714	0.9817
							simulation	0.3333	0.8333

Another algorithm for FFT in multiprocessors is the shuffling FFT. In this algorithm, computations of partial FFTs alternate with shuffling stages in which data are passed among processors. Only two processors can share an S-block. Figure 6 presents the shuffling FFT algorithm for an example where N=16 and P=4. There is much more locality in this algorithm. Coherence activity is much reduced, making this algorithm more suitable for cache-based systems.

During each butterfly computation and each shuffling stage, each shared block may be read and updated by one processor only. There is only one set of shared writable data in this algorithm and each data is shared by two processors. Table 5 shows the results for P=16. The model is in good agreement with the exact values (obtained analytically and verified by simulation.) There is less shared data contention than in the non-shuffling FFT.

Table 5: Shuffling FFT algorithm (P=4,16 and 64K data elements)

P	J	W	l	f	p_s	n_s		miss	penalty
4	2	0.600	17.8	0.667	0.7500	49152	model	0.0158	0.0363
							simulation	0.0171	0.0426
16	2	0.600	15.4	0.667	0.9375	61440	model	0.0228	0.0525
							simulation	0.0247	0.0617

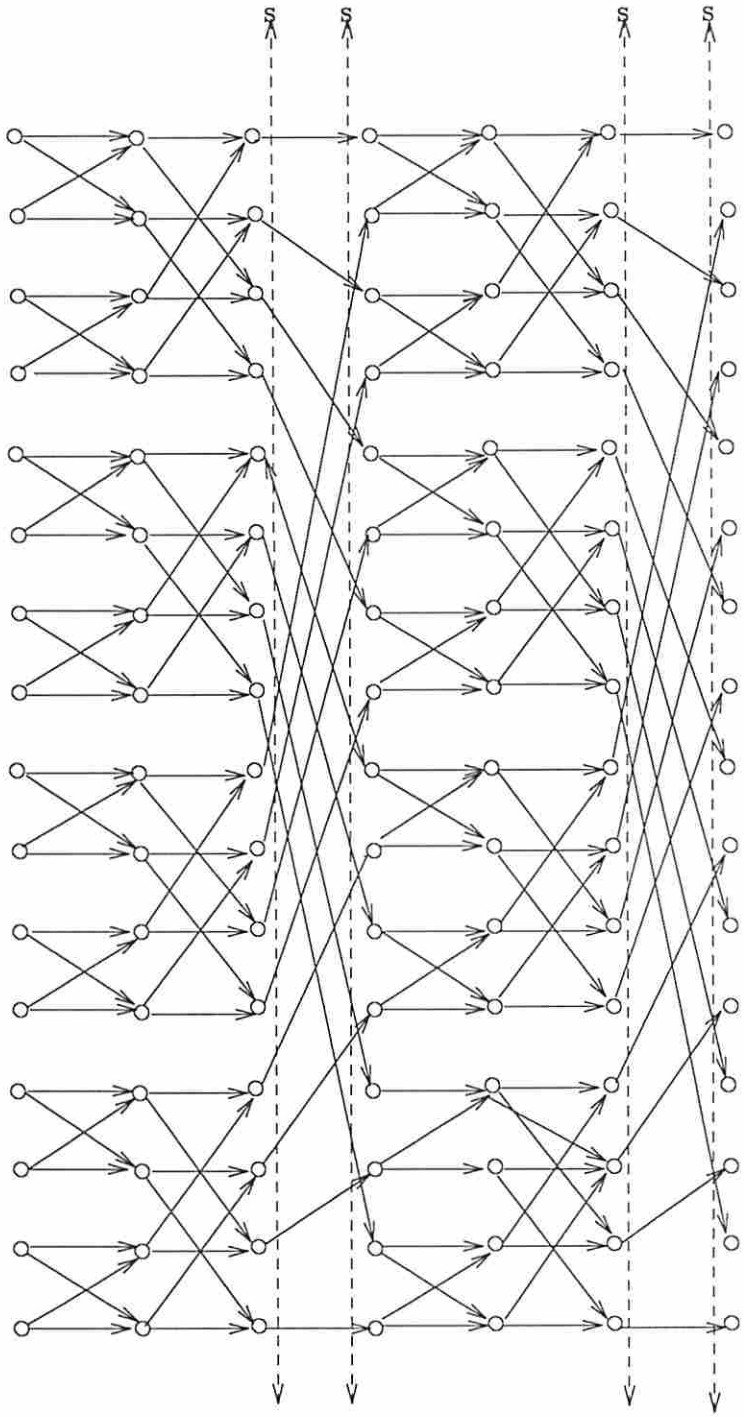


Figure 6: Shuffling FFT algorithm for P=4 and N=16.

5.4 System Effects

Table 6 records the overall miss rate on data and the average coherence penalty for the five algorithms. There are three numbers per entry in the Table. The first number is the sum of the contributions of each set calculated from the analytical model. The second number is obtained by computing the average values of the model parameters and by applying these average values to the model. These two numbers are very close because there is only one set of data, or one set dominates, or all sets are accessed with the same probability. The third number is obtained by simulation. In the case of the quicksort algorithm, the simulations were run for ten independent 32K random files, for P=2,4 and 8. Averages were taken for all analytical model parameters and for the penalty and miss rate resulting from simulations.

These performance estimates apply to the part of the algorithm where sharing of writable data occurs; moreover, the first miss and its associated penalty are not counted.

6. EFFECT OF CACHE BLOCK SIZE

The cache block size is an important factor affecting the system performance. When a cache block contains more than one datum, the sets of S-blocks are very different and, in general, it is much more difficult to apply the models.

In this section, we revisit the five multitasked algorithms discussed in the previous section, and show how the parameters, J, W, l, and f are affected by the cache block size, B.

In the following, we chose the following penalty rate for each coherence event: $\lambda_M = \lambda_{CS\_RW} = \lambda_{IN\_RW} = 0.75 + 0.25B$ and $\lambda_{IN\_RO} = 0.5$. Events CS_RW and IN_RW require two block transfers in our protocol: one to write the block to main memory and one to transfer the block to the cache. We model the penalty for a block transfer by a simple linear function of the block size.

6.1 Relaxation Algorithms for Partial Differential Equations

In the two iterative algorithms, when the cache block size increases, the number of sets of S-blocks also increase. For instance, there are five sets of S-blocks when B is two, eight sets of S-blocks when B is four, and ten sets of S-blocks when B is eight.

Take for example the Jacobi iterative algorithm; let's consider a cache block size of four data elements and a grid size of 128 $\times$ 128. Arrays are stored row-wise. Figure 7 illustrates the eight different types of S-blocks, named type 1 to type 8. Type 1 S-blocks are read $4B$ times in two successive iterations (each of these accesses is counted as a burst of size 1); besides these multiple Read accesses, there are two consecutive Write bursts in which the block is modified a total of B times; hence, the parameter W of type 1 S-block is $\frac{2}{4B+2}$;

Table 6: System Effects (block size of one)

(1) Model using average values of the parameters
(2) Sum of the contributions of each set (model)
(3) Sum of the contributions of each set (simulation)
In the following Table, parameters, J, W, l, f, are average values.

Algorithm	J	W	l	f	q_s	N_s		miss	penalty
Jacobi iteration	2.01	0.201	1	1	0.0309	1016	(1)	0.0052	0.0125
(P=4)							(2)	0.0052	0.0124
128 X 128 grid size							(3)	0.0063	0.0156
S.O.R. iteration	2.01	0.201	1.201	0	0.0309	508	(1)	0.0043	0.0108
(P=4)							(2)	0.0043	0.0104
128 X 128 grid size							(3)	0.0053	0.0130
Quicksort	1.97	0.486	6.878	0	1.0000	32768	(1)	0.0465	0.1170
(P=2)							(2)	——	——
							(3)	0.0767	0.1895
(P=4)	3.32	0.548	5.575	0	1.0000	32768	(1)	0.1003	0.2195
							(2)	——	——
							(3)	0.1156	0.2697
(P=8)	4.58	0.568	5.024	0	1.0000	32768	(1)	0.1355	0.2799
32K data elements							(2)	——	——
							(3)	0.1389	0.3150
Non-shuffling	3.00	0.333	1	1	1.0000	65536	(1)	0.4000	0.7810
FFT							(2)	0.4000	0.7810
(P=4)							(3)	0.3333	0.8333
(P=16)	5.00	0.333	1	1	1.0000	65536	(1)	0.5714	0.9817
64K data elements							(2)	0.5714	0.9817
							(3)	0.3333	0.8333
Shuffling FFT	2.00	0.600	17.8	0.667	0.7500	49152	(1)	0.0158	0.0363
(P=4)							(2)	0.0158	0.0363
							(3)	0.0171	0.0426
(P=16)	2.00	0.600	15.4	0.667	0.9375	61440	(1)	0.0228	0.0525
64K data elements							(2)	0.0228	0.0525
							(3)	0.0247	0.0617

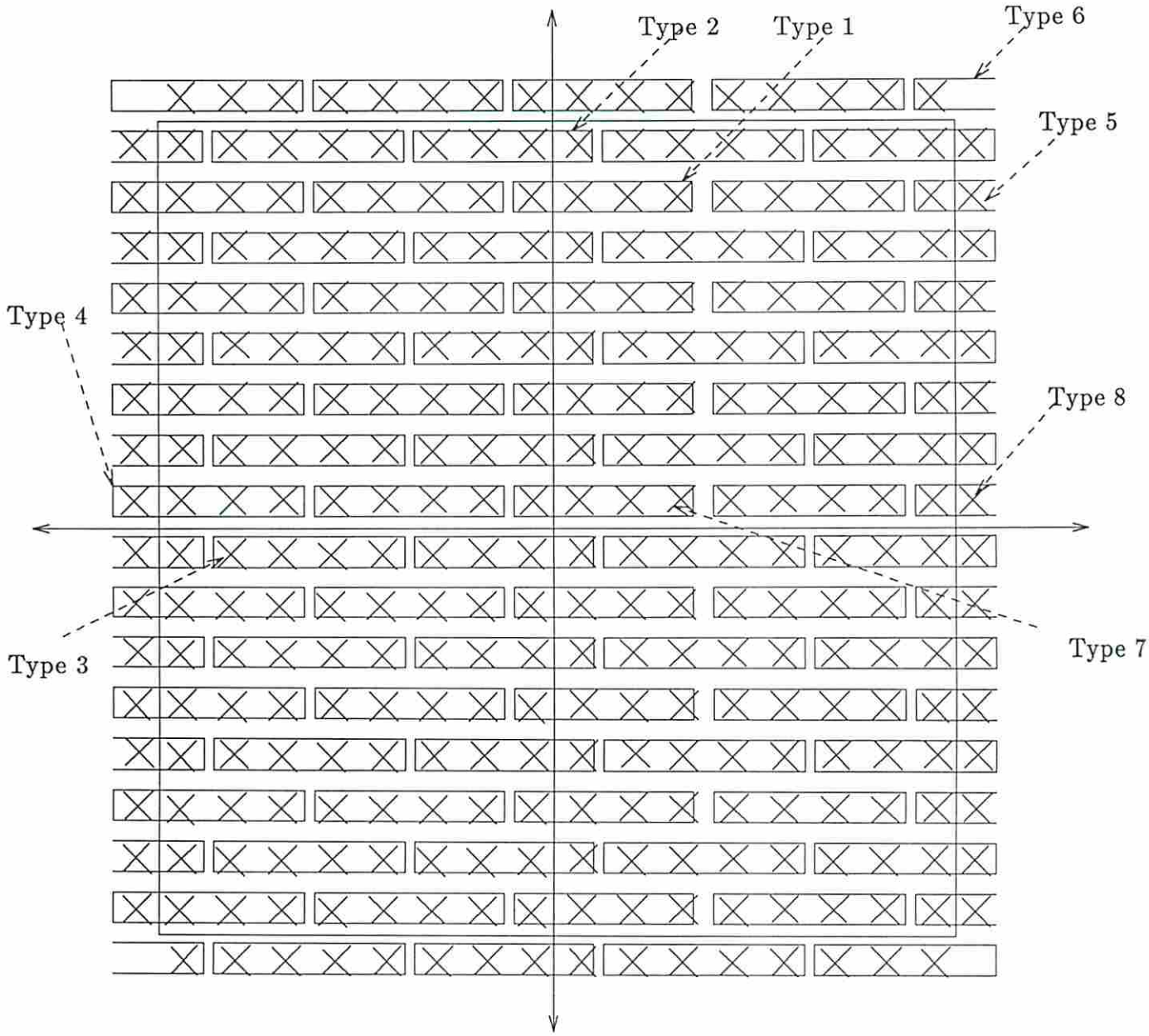


Figure 7: The eight sets of S-blocks in the Jacobi iteration when B is equal to four.(M=16, P=4)

the average number of accesses in a burst, l, is equal to $\frac{5B}{4B+2}$. Parameter f is equal to one since there are only Write operations in the Write bursts. Similarly, parameters of type 2 to type 8 S-blocks can be computed. Table 7 lists all parameters for the eight different types of S-blocks. Note that this Table is only valid for B=4. For each value of B, a different table will be needed.

Table 7: Parameters for Jacobi Iteration (B=4)

Set	J	W	l	f
type 1	2	$\frac{2}{4B+2}$	$\frac{5B}{4B+2}$	1
type 2	2	$\frac{2}{3B+2}$	$\frac{4B}{3B+2}$	1
type 3	2	$\frac{1}{4B+1}$	$\frac{5B}{4B+1}$	1
type 4	2	$\frac{1}{4(B-1)+1}$	$\frac{5B-5}{4B-3}$	1
type 5	2	$\frac{2}{[4(B-2)]+2}$	$\frac{5B-10}{4B-6}$	1
type 6	2	$\frac{1}{4(B-3)+1}$	$\frac{9B-26}{8B-22}$	1
type 7	4	$\frac{2}{4B+2}$	$\frac{5B}{4B+2}$	1
type 8	4	$\frac{2}{[4(B-2)]+2}$	$\frac{5B-10}{4B-6}$	1

In the case of an $M \times M$ Jacobi array, when the cache block size exceeds $\frac{M}{\sqrt{P}}+1$, S-blocks moves back and forth between processors in the Write phase, that is, processors update S-blocks alternatively, and hence the number of references l in each burst is equal to one; J can be as high as $2 \cdot \sqrt{P}$ or even P, $W = \frac{1}{5}$, and f=1.

We have applied the above analysis for block sizes from 1 to 256, and for a grid size of 128 $\times$ 128. Figures 8 and 9 show the comparison between model prediction(dotted curve) and simulation (plain) for the system miss ratio and the system penalty (obtained by summing the contributions of all sets). These curves are valid for any number of processors P provided $B < \frac{M}{\sqrt{P}}+1$. These two Figures show that the analytical program model can make very good predictions for the Jacobi iteration algorithm when the cache block size is greater than two (error is less than 5%).

In the S.O.R. algorithm, for any cache block size, the number of sets of S-blocks are the same as in the Jacobi iteration algorithm; however, the reference patterns to the blocks in the sets are different. In the case of an $M \times M$ S.O.R. array, when the cache block size exceeds $\frac{M}{\sqrt{P}}+1$, S-blocks are updated alternatively by different processors. In this case, J can bee as high as $2 \cdot \sqrt{P}$ or even P, $W = \frac{1}{6}$, l=1, and $f = 1$.

Figures 10 and 11 illustrate the results of the system miss ratios and system penalities for different cache block sizes for a 128 $\times$ 128 grid. These curves are independent of the number of processors provided $B < \frac{M}{\sqrt{P}}+1$. The model (dotted) is compared to the

simulation (plain). The Figures show that the model is very reliable in this case.

6.2 Quicksort

Results are shown for an eight-processor system with data file size of 32,768. The values of the parameters are obtained from the simulator and are average values. Figures 12 and 13 show the system miss ratio and the system penalty obtained from the model (dotted) and from the simulation (plain). Each of the simulated point is an average result of simulation runs for ten independent random files. Even though the relative error in these two Figures is large when the cache block size is greater than four, the model is very good at predicting the general trend.

6.3 Fast Fourier Transform (FFT)

In the non-shuffling FFT algorithm, there is only one set of S-blocks. When the number of data elements in a cache block is less than or equal to $\frac{N}{P}$, each S-block is shared by $log_2 P + 1$ processors; there are $2B$ isolated Reads followed by one Write burst with B Writes. Therefore, J is equal to $log_2 P + 1$, l is equal to $\frac{3B}{2B+1}$, W is equal to $\frac{1}{2B+1}$, and f is equal to 1. When the number of data elements in one cache block exceeds $\frac{N}{P}$, S-blocks bounce back and forth between processors at every Write.

In the shuffling FFT algorithm, there is also only one set of S-blocks. When the number of data elements in a cache block is less than or equal to $\frac{N}{P}$, each S-block is shared by two processors. Write operations always occur in the butterfly computation phases and never occur in the shuffling phases. When the number of data elements in one cache block is larger than $\frac{N}{P}$, S-blocks move back and forth between processors.

Figures 14-17 show the results for the non-shuffling and shuffling FFT algorithms. The file size is 64K and the number of processors is four. These curves would be different but would have the same shape for larger number of processors. The relative errors between model predictions and simulations are between 20% and 30% for the system miss ratio, and between 5% and 20% for the system penalty.

7. Finite Cache Effects

When can a cache be considered as infinite for a set of blocks? The answer mostly depends on the number of additional misses caused by replacement in a finite cache system. If this number of misses is small compared to the invalidation rate, then its effect on the coherence overhead can be neglected. To clarify this issue, we have run multiple simulations for systems with finite caches, and we have compared them to the results for the infinite cache system.

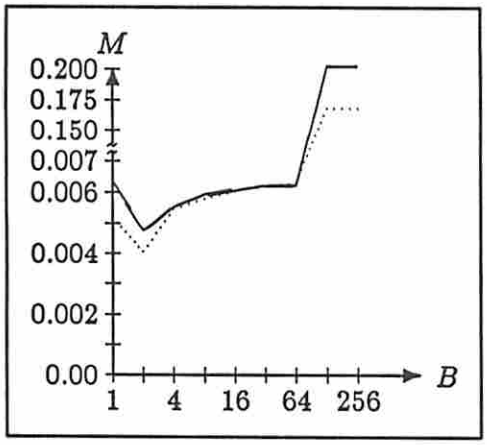


Figure 8: The system miss ratio for the Jacobi iterative algorithm

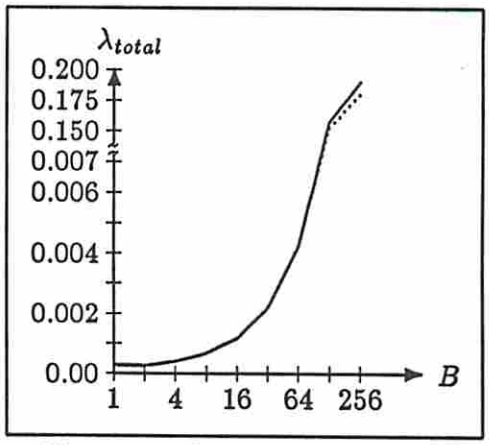


Figure 9: The system total penalty for the Jacobi iterative algorithm

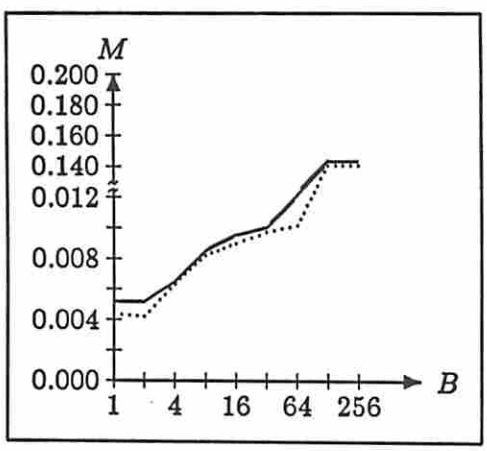


Figure 10: The system miss ratio for the S.O.R. iterative algorithm

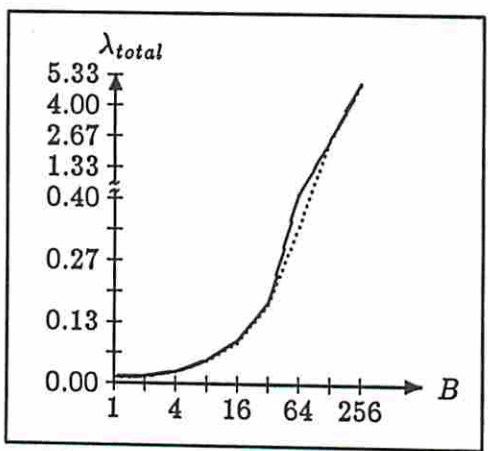


Figure 11: The system total penalty for the S.O.R. iterative algorithm

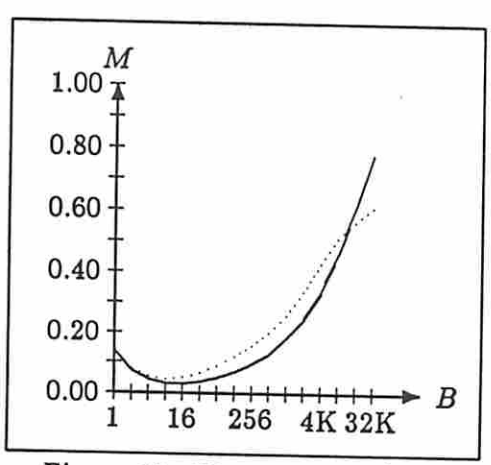


Figure 12: The system miss ratio for the quicksort algorithm ($P = 8$)

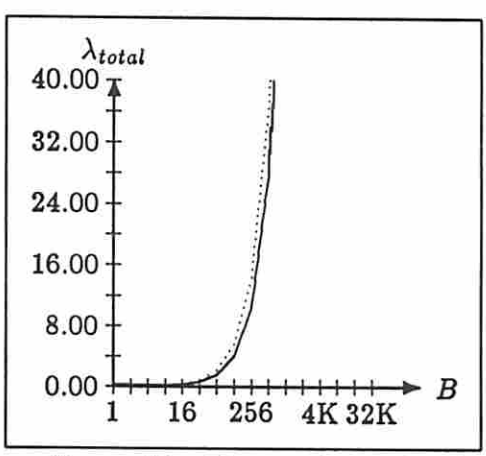


Figure 13: The total penalty for the quicksort algorithm ($P = 8$)

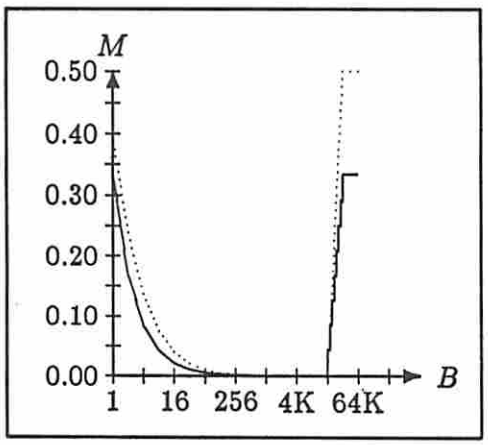


Figure 14: The system miss ratio for the Non-shuffling FFT algorithm

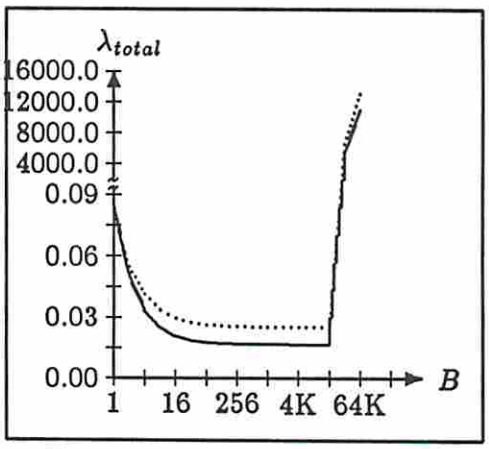


Figure 15: The system total penalty for the Non-shuffling FFT algorithm

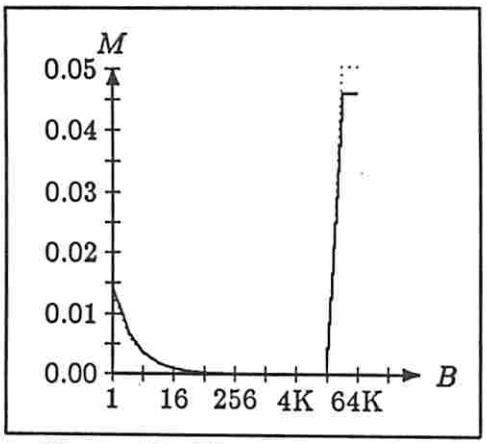


Figure 16: The system miss ratio for the Shuffling FFT algorithm

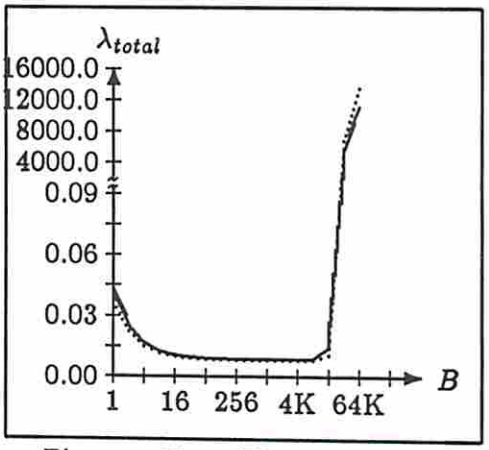


Figure 17: The system total penalty for the Shuffling FFT algorithm

solid line : simulation result
dash line : model prediction

In the finite cache system, a new penalty,$\lambda_{write-back}$, must be defined. Since a write-back involves a block transfer, $\lambda_{write-back} = \lambda_M$ in the example. In the simulations, the caches are fully associative with an LRU replacement policy and the block size B=4. Finite cache effects for the S.O.R.iterative algorithm are displayed in Figures 18 (for M) and 19 (for λ_{total}). In these simulations, the cache size is varied from 512 to 64K words. A detailed study of the impact of the cache size in the performance of the S.O.R. algorithm was also done in [9]. As the cache size increases, the miss ratio and the average penalty decrease, until the caches are so big that all the data accessed during the whole execution of the algorithm can be contained in them. At this point there is no replacement. This behavior is typical of the five algorithms in this paper. The results for the ideal system with infinite caches are therefore lower bounds and they can be reached provided the caches are large enough. Studying finite cache effects in details is very complex because of the number of parameters involved and the length of each simulation run.

8. DISCUSSION OF RESULTS

It has been observed that the combined effects of critical sections (for all block sizes) and of the spatial locality [19] of accesses (for block sizes larger than one) to shared writable blocks result in access bursts to such blocks by different processors. This is the basic premise of the paper. Based on this observation, we have extended a previous program model for the sharing of data, and we have tried to match the model predictions and the predictions of simple simulations of algorithms in multiprocessors with infinite caches.

It appears that iterative algorithms such as the Jacobi or S.O.R. are very well suited to cache-based systems with large data caches, because shared data contention is low (in realistic cases, the number of processors sharing a given writable block is less than four and the fraction of accesses to shared writable data is low). Figures 9-12 show that bigger block sizes do not improve the overall hit rate on shared data and cause more penalty: the average miss rate on each S-block access decreases (i.e., M_i decreases) but the number of accesses to such blocks increases (i.e., q_s increases); the probability of a coherence event per access to S-blocks decreases, but this is more than compensated by the increase of q_s and of the penalty associated with each coherence event. For shared data accesses, the block size should be small (one or two data elements). Note that this conclusion is only valid if the caches are large enough to contain all data across successive iterations; the first iteration causes large number of misses for the initial load of data and instructions. These transients are helped by a bigger block size. From the Figures, we observe that a block size of 16 data elements is acceptable for shared data accesses. These conclusions are valid for many configurations of processors as explained in Section 6.1.

The results of quicksort are the average values of results of multiple random input files. Bigger block sizes do not improve the performance of the quicksort algorithm when cache block size is greater than eight since large cache blocks have higher probability to be shared by multiple processors.

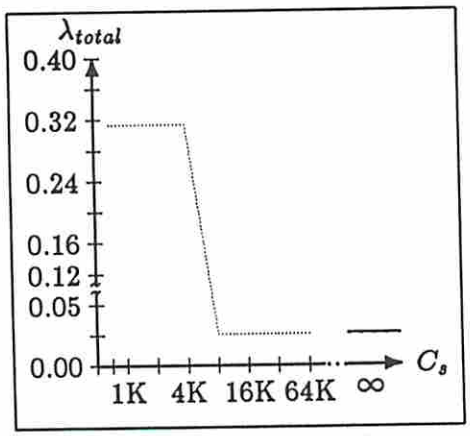


Figure 19. Cache size effects on λ_{total} for the S.O.R. iterative algorithm

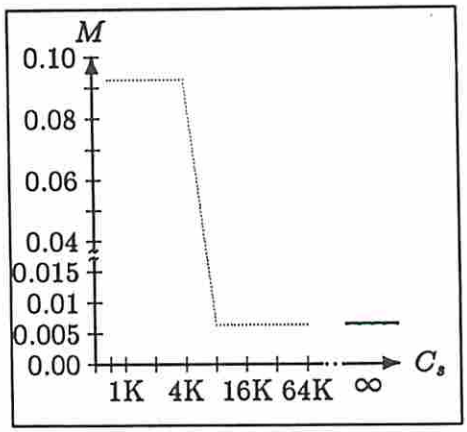


Figure 18. Cache size effects on M for the S.O.R. iterative algorithm

solid line : infinite cache simulation result
dash line : infinite cache model prediction
dotted line : finite cache simulation result

Bigger block sizes also improve the performance of the FFT routines (strangely enough up to a block size of 16 data elements, again). While the penalties on individual coherence events increase with the block size, the number of shared data accesses causing these events decreases, as the block size increases.

In all the simulations we ran, there is a maximum block size beyond which the performance drops sharply. This block size depends on the size of the problem and on the decomposition of the algorithm (i.e., the number of processors).

From Tables 1 and from Figures 9-18, it appears that for the five algorithms studied in this paper, the precision of the model based on the idea of access bursts is good in many cases. We never expected the models to fit exactly each case: because of the large data reduction in the stochastic models, a given model with given parameter values maps on different algorithms with different behaviors. It appears however that the models and their parameters are sufficient to approximate the shared data contention effect for some important parallel algorithms, and in the case of the infinite cache model.

If we look at the comparisons between model and simulations, it appears that the quicksort and the non-shuffling FFT result in the worst predictions; in the case of the non-shuffling FFT, the model predicts that the coherence overhead will increase with P, the number of processors, while the simulations predict that it remains constant. A closer look at Figure 6 shows that an S-block is not shared by all $log_2 P + 1$ processors at all times but rather that it is shared by different groups of two processors at different stages of the computation. Applying the model with J=2 would yield a much improved prediction of the model.

We have assumed all along that preemptions and migrations of processes were disallowed. We mostly made this assumption to simplify the analysis of the algorithms. However, in many cases, the bursty behavior of accesses would be preserved if preemption and migration were allowed. The reason is that bursts of accesses are short and unlikely to be interrupted by preemption. Migration would increase the randomness in the selection of the processor to start the next access burst, and therefore would increase the accuracy of the model for the non-shuffling FFT. While the parameters W, f and l would remain the same as in this paper (assuming that time between preemptions is very large compared to the average burst time), the parameter J would have to be different. If migration is allowed, then private writable blocks will behave like shared writable blocks and the model could be applied to private blocks, as well.

9. CONCLUSIONS

In this paper, we have presented and solved a simple model for the caching of shared writable data in multiprocessor systems executing parallel algorithms. The simplicity and generality of the results stem from the infinite cache hypothesis. The infinite cache model is independent of all cache parameters (e.g, organization or replacement policy).

There are many extensions possible to this work. First of all, one could analyze the

behavior of more parallel algorithms and compare it to the model predictions. One could investigate the effect of migration, preemption or dynamic scheduling. Some parameters in the model, such as l, are easier to estimate directly than others [13], such as J, when migration and preemption are allowed. Besides using the results of simulations or measurements to estimate these parameters, one can use the model to derive upper bounds (for example, if $J=P$ or if $W=1$, then the miss ratio obtained through equation (4) is an upper bound.) Finite cache effects should be studied in detail.

Finally, given the simplicity of the program behavior models, one can derive simple results for proposed coherence protocols in order to compare their effectiveness in handling shared writable blocks [22].

References

[1] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. An evaluation of directory schemes for cache coherence. In *Proceedings of 15th Annual International Symposium on Computer Architecture*, pages 280–289, June 1988.

[2] A.O. Allen. *Probability, Statistics, and Queueing Theory.* Academic Press, 1978.

[3] G.R. Andrews and F.B. Schneider. Concepts and notations for concurrent programming. *Computing Surveys*, 15(1), March 1983.

[4] J. Archibald and J.L. Baer. Cache-coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4(4):273–298, November 1986.

[5] F.A. Briggs and M. Dubois. Effectiveness of private caches in multiprocessor systems with parallel-pipelined memories. *IEEE Transactions on Computers*, C-32(1), January 1983.

[6] P.J. Courtois, F. Heymans, and D.L. Parnas. Concurrent control with readers and writers. *Communications of the ACM*, 14(10):667–668, October 1971.

[7] Z. Cvetanovic. The effect of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Transactions on Computers*, C-36(4), April 1987.

[8] M. Dubois. Effect of invalidations on the hit ratio of cache-based multiprocessors. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 255–257, August 1987.

[9] M. Dubois. Throughput analysis of cache-based multiprocessors with multiple buses. *IEEE Transactions on Computers*, C-37(1):58–70, January 1988.

[10] M. Dubois and F.A. Briggs. Effects of cache coherency in multiprocessors. *IEEE Transactions on Computers*, C-31(11):1083–1099, November 1982.

[11] M. Dubois, F.A. Briggs, I. Patil, and M. Balakrishnan. Trace-driven simulations of parallel and distributed algorithms in multiprocessors. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 909–916, August 1986.

[12] M. Dubois and J.C. Wang. Shared data contention in a cache coherence protocol. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 146–155, August 1988.

[13] M. Dubois and J.C. Wang. *Shared Data Contention in a Cache Coherence Protocol.* Technical Report CRI 88-26, CRI Department of Electrical Engineering, University of southern California, 1988.

[14] S.J. Eggers and R.H. Katz. A characterization of sharing in parallel programs and its application to coherency protocol evaluation. In *Proceedings of 15th Annual International Symposium on Computer Architecture*, pages 373–382, June 1988.

[15] K. Hwang and F.A.Briggs. *Computer Architecture and Parallel Processing.* Mac Graw-Hill, 1984.

[16] H.T. Kung. *Algorithms and Complexity: New Directions and Recent Results.* J.F. Traub Ed., New York: Academic Press, 1976.

[17] R.L. Lee, P-C Yew, and D.H. Lawrie. Multiprocessor cache design considerations. In *Proceedings of 14th Annual International Symposium on Computer Architecture*, June 1987.

[18] J.H. Patel. Analysis of multiprocessors with private cache memories. *IEEE Transactions on Computers*, C-31(4):296–304, April 1982.

[19] A.J. Smith. Cache memories. *Computing Surveys*, 14(3):473–530, September 1982.

[20] J.R. Spirn. *Program Behavior : Models and Measurements.* Elsevier Computer Science Library, 1977.

[21] M.K. Vernon and M.A. Holliday. Performance analysis of multiprocessors cache consistency protocols using generalized timed petri nets. In *Proceedings of the Performance 1986 and ACM Sigmetrics 1986 Joint Computer Performance Modeling, Measurement and Evaluation*, pages 9–17, May 1986.

[22] J.C. Wang and M. Dubois. A performance comparison of cache coherence protocols based on the access burst model. In *Proceedings of the Second Annual Parallel Processing Symposium*, pages 73–87, April 1988.

[23] D. Young. *Iterative Solution of Large Linear Systems.* Academic Press: New York, 1971.

APPENDIX A Miss ratio derivation

To facilitate the derivation of the formula for the miss rate, we drop the index i in all equations. Let's define

$$X_1 = \sum_{k=1}^{J} k \cdot Pr(k)$$

and

$$X_2 = \sum_{k=1}^{J} k^2 \cdot Pr(k) \ .$$

After multiply both sides of equation (1) of Section 4.1, we obtain

$$k \cdot [J - k \cdot (1 - W)] \cdot Pr(k) = k \cdot [J - (k-1)] \cdot (1 - W) \cdot Pr(k-1)$$

or

$$\begin{aligned} k \cdot [J - k \cdot (1 - W)] \cdot Pr(k) = \ & \{ \ (k-1) \cdot [J - (k-1)] \cdot (1 - W) \\ & + \ [J - (k-1)] \cdot (1 - W) \} \cdot Pr(k-1) \end{aligned}$$

or

$$[k \cdot J - k^2 \cdot (1 - W)] \cdot Pr(k) = (1 - W) \cdot [J + (J-1) \cdot (k-1) - (k-1)^2] \cdot Pr(k-1)$$
$$for \quad k = 2, ..., J \ .$$

Let's sum all the left hand sides of these equations (from k=2 to J) and let's equate the result to the sum of the right hand sides; we have

$$\begin{aligned} & J \cdot X_1 - J \cdot Pr(1) - (1 - W) \cdot X_2 + (1 - W) \cdot Pr(1) \\ = \ & J \cdot (1 - W) \cdot (1 - Pr(J)) + (J - 1) \cdot (1 - W) \cdot (X_1 - J \cdot Pr(J)) \\ & -(1 - W) \cdot (X_2 - J^2 \cdot Pr(J)) \ . \end{aligned}$$

After some simplifications, we have

$$\begin{aligned} [J - (J-1) \cdot (1 - W)] \cdot X_1 \ &= \ J \cdot (1 - W) + (J - 1 + W) \cdot Pr(1) \\ &= \ J \end{aligned}$$

or

$$X_1 = \frac{J}{1 + (J-1) \cdot W} \ .$$

Since, from equation (3) in Section 4.1,

$$1 - \frac{X_1}{J} = \sum_{k=1}^{J} \frac{J-k}{J} \cdot Pr(k) = l \cdot M$$

it follows that

$$M = \frac{1}{l} \cdot \frac{(J-1) \cdot W}{1 + (J-1) \cdot W} \ . \qquad (QED)$$

APPENDIX B

In this appendix, we will explain how we have obtained the values of the model parameters and the values of the actual miss rate and coherence penalties for the five algorithms. The actual miss rate and coherence penalties have been verified through simulation by using the tool described in [11]. This appendix also assumes that a cache block contains only one data element and the initial miss is never counted.

1. Jacobi iteration
   There are 3 sets of S-blocks in the Jacobi iteration algorithm. Parameters of each set can be computed as follows.

   (a) For the inner grid points with J=2, the total number of accesses in the multiple Read phase is four and in the next iteration (critical section phase), the total number of accesses is one Write. Therefore, the values of the parameters are J=2, l=1, and $W=\frac{1}{5}$.

   (b) For the outer grid points with J=2, the reference pattern is very similar to that of the previous set except that the total number of accesses in the multiple Read phase is three since the boundary points need not be updated during the execution. Hence, the values of the parameters are J=2, l=1, and $W=\frac{1}{4}$.

   (c) For the inner grid points with J=3, the behavior is exactly the same as that of the set of inner grid points with J=2. Thus, the values of the parameters are J=3, l=1, and $W=\frac{1}{5}$.

   The actual miss rate and coherence penalties can also be easily found in the Jacobi iteration algorithm because of the simplicity of the algorithm. Each of them is shown below.

   (a) For the inner grid points with J=2, let's take the block i shared by P0 and P1 as an example. After every multiple Read phase, both caches of P0 and P1 have the block i. In the next iteration (critical section phase), one of the processor, say P0, updates the block i, invalidates the copy in the cache of P1, and sets the state of block i to 1_RW; since the total number of references for both iterations are five, I is $\frac{1}{5}$. In the next multiple Read phase, a miss occurs when P1 references block i and changes its state to 2_RO; thus M and C are also $\frac{1}{5}$.

   (b) The reference pattern to the outer grid points with J=2 are the same as that to inner grid points with J=2 except that there are three Reads in the multiple Read phase. Hence, M, I, and C are equal to $\frac{1}{4}$.

(c) The inner grid points with J=3 can be shared by three processors, P0, P1, and P2. In every Write phase, P0 modifies a block j, invalidates the copy of the block in the caches of P1 and P2, and sets the state of the block to 1_RW; I is therefore equal to $\frac{1}{5}$. In every multiple Read phase, two misses occur when P1 and P2 try to read the block and the block j stays in state 3_RO after this phase. Hence, M=$\frac{2}{5}$ and C=$\frac{1}{5}$.

There is no D event in the above. Also, since there is only one Write access in the critical section, f is equal to one in all three sets.

2. S.O.R. iteration
The sets of S-blocks in the S.O.R. algorithm are the same as in the Jacobi iteration. The reference pattern to elements of each set are identical in the two algorithms, except that in the S.O.R. algorithm the data is both read and written in the critical section phase. Thus, the values of the parameters and actual miss rate and coherence penalties can be written as follows.

   (a) For the inner grid points with J=2, we have J=2, l=$\frac{6}{5}$, W=$\frac{1}{5}$, M=$\frac{1}{6}$, I=$\frac{1}{6}$, C=$\frac{1}{6}$, and D=0.

   (b) For the outer grid points with J=2, we have J=2, l=$\frac{6}{5}$, W=$\frac{1}{4}$, M=$\frac{1}{5}$, I=$\frac{1}{5}$, C=$\frac{1}{5}$, and D=0.

   (c) For the inner grid points with J=3, we have J=3, l=$\frac{6}{5}$, W=$\frac{1}{5}$, M=$\frac{2}{6}$, I=$\frac{1}{6}$, C=$\frac{1}{6}$, and D=0.

   Since, in the critical section, each Write access is preceded by a Read access, f is equal to zero for each of the three sets.

3. Non-shuffling FFT algorithm
We have derived the model parameters for the non-shuffling FFT algorithm in section 5.3. Here, we only compute the actual miss rate and coherence penalties for the algorithm.

   In the non-shuffling FFT algorithm, every S-block is accessed twice in the multiple Read phase and only once in the critical section phase. Like the Jacobi and S.O.R. algorithms, a copy of an S-block is invalidated in the critical section phase, and a miss and a CS_RW event occur in the multiple Read phase. Therefore, the actual miss rate on an S-block, the probability of event IN_RO and the probability of event CS_RW are all $\frac{1}{3}$, independent of P and J.

4. Shuffling FFT algorithm
There is only one set of shared writable data in the shuffling FFT algorithm and each data is shared by two processors, that is, J=2. The total number of butterfly/shuffling stages is $2 \lceil \frac{log_2 N}{log_2 \frac{N}{P}} \rceil$. To compute the parameters of the program model,

we note that Writes always occur in the butterfly computation phases and never occur in the shuffling phases. Because of the alternation between Reading and Writing phases,

$$W = \frac{\lceil \frac{log_2 N}{log_2 \frac{N}{P}} \rceil + 1}{2 \lceil \frac{log_2 N}{log_2 \frac{N}{P}} \rceil + 1}$$

where the '1' in the formula corresponds to the Write operation at the end (see Figure 6). In each butterfly computation, there are $(3\ log_2 \frac{N}{P} + 1)$ accesses to the block and in the shuffling phase there is only one Read. Thus

$$l = \frac{(3\ log_2 \frac{N}{P} + 2) \lceil \frac{log_2 N}{log_2 \frac{N}{P}} \rceil + 1}{2 \lceil \frac{log_2 N}{log_2 \frac{N}{P}} \rceil + 1}$$

In the shuffling FFT algorithm, an event CS_RW and a miss occur every time a processor reads an S-block from a different processor's cache in the shuffling stage; both processors then own the S-block as RO copy. These events are followed by an IN_RO event in the butterfly computation stage when a processor update the S-block. Thus, the value of M, C, and I are identical and equal to

$$\frac{\lceil \frac{log N}{log \frac{N}{P}} \rceil}{(3\ log \frac{N}{P} + 2) \lceil \frac{log N}{log \frac{N}{P}} \rceil}$$

where $(3\ log \frac{N}{P} + 2) \lceil \frac{log N}{log \frac{N}{P}} \rceil$ is the total number of references to an S-block. The probability of event IN_RW is zero for both non-shuffling and shuffling FFT algorithms because there is no Write operation when a shared block is in the state 1_RW. Parameter f for non-shuffling FFT algorithm is one since there is only one Write access in the critical section; however, parameter f for shuffling FFT algorithm is $\frac{\lceil \frac{log N}{log \frac{N}{P}} \rceil - 1}{\lceil \frac{log N}{log \frac{N}{P}} \rceil}$ since the first operation in the butterfly computation stage is always a Write except the first butterfly computation stage, and there is only one Read operation in a shuffling stage.

PART II: A PERFORMANCE COMPARISON OF CACHE COHERENCE PROTOCOL BASED ON THE ACCESS BURST MODEL

Abstract

The choice of a cache coherence protocol is an important design consideration in multiprocessor systems. Several protocols have been proposed and techniques must be developed to compare their relative merits in different computing environments.

In this paper, we compare five write-invalidate protocols for their effectiveness in handling shared writable blocks. A program model for data sharing called the access-burst model is applied to the different protocols. This model takes into account the fact that shared writable data are accessed in critical and semi-critical sections.

Protocols are modeled by Markov chains; an analytical closed-form solution is derived for all components of the cache coherence overhead and for all cache coherence protocols in systems with caches of infinite sizes.

Index term: cache coherence protocols, program model, trace-driven simulations, multiprocessors, multitasking

1 Introduction

The goal of current research in parallel processing is to run concurrent and cooperating processes on several processors in order to minimize the execution time of an algorithm. Parallel processes need to access shared writable data to communicate and synchronize. In a multiprocessor system, when each processor has a private cache memory, several copies of the same cache block may exist in different cache memories at the same time; therefore a mechanism must exist to enforce the consistency of all copies of the same block. Different cache coherence protocols capable of maintaining the consistency of shared data have been discussed in [3].

"Snooping" cache coherence protocols are applicable to bus-based systems, because each cache in the system must observe all of the coherence transactions. A cache must respond to a coherence transaction on the shared memory bus for each block present in its directory.

Snooping cache protocols can be classified into two categories, *write-invalidate* protocols [5,11,12,13,15], and *write-broadcast* protocols [14,16]. The first type of protocols maintain consistency by invalidating all copies in other caches on a write. The Basic [8], the Write-once [12], the Synapse [11], the Illinois [15] and the Berkeley [13] cache coherence protocols fall into this category. In write-broadcast protocols such as the Firefly [16] and the Dragon [14] protocols copies are updated instead of being invalidated.

In this paper, we apply an analytical program model called the *access burst model* [9], to compare the effectiveness of different coherence protocols in handling shared writable data. The access burst model was introduced in [9], and is based on the observation that shared writable data are accessed in critical or semi-critical sections [8]. Its predictions were compared with trace-driven simulation results of five algorithms, for the Basic coherence protocol.

Caches have infinite sizes and models are derived for computations in steady-state. This simplification drastically reduces the number of parameters in the models. The results obtained are an indication of protocol efficiency for very large caches and compute-intensive, iterative algorithms. Many such algorithms exist for asynchronous multiprocessors [4]. An example of such an algorithm is given at the end of the paper. These two restrictions on the system were also assumed in the paper by Eggers and Katz [10] and in the paper by Agarwal *et al.* [1], in which some trace-driven simulations are presented, but no analytical model is proposed.

The remainder of this paper is organized as follows. Section 2 briefly describes the access burst model. Section 3 applies the model to analyze the five write-invalidate protocols. Section 4 compares the prediction of the model with trace-driven simulations in the case of a specific algorithm. Finally, the protocols are compared in the light of the models in Section 5.

2 Program Model

In multiprocessor systems, we distinguish between two classes of shared variables, namely, synchronization variables and shared writable operands, accessed in *critical sections* [2] (only one process can access the data at a time either on a Read or on a Write) or in *semi-critical sections* [6] (multiple processes can read a data item at the same time, but only one process can modify the data item at a time). If a block can be read and modified by different processors then we call the block an S-block; otherwise, it is a P-block. Accesses to synchronization variables are not considered in this paper.

The P processors generate homogeneous streams of references to P- and S-blocks. S-blocks may belong to different *sets*, based on the reference patterns. In any multitasked implementation of an algorithm, it is possible to identify sets of S-blocks shared by given groups of processors. *In the following, we analyze the coherence overhead for a given set of S-blocks.* In practice, the contributions of each set must be added [9]. A great advantage of the infinite cache assumption is that a set of S-blocks can be analyzed in isolation. Only the access pattern to the S-blocks in the set need be specified. Also, the infinite cache result for a given set of S-blocks is independent of all cache parameters besides the block size.

While a processor accesses a shared writable datum, no other processor is able to *interleave* accesses to the same datum. Accesses to a shared datum by one processor therefore occur in uninterrupted bursts. When a cache block can contain more than one data element, the locality of accesses also contributes to access bursts. If $q_s \cdot p_i$ is the fraction of references to an S-block i shared by J processors and if ls is the average number of references to S-block i in each access burst – when the block size is one datum, then ls is also the number of accesses to the datum in the critical section– the probability of starting an access burst for S-block i is $q_s \cdot p_i / ls$. In the semi-critical section model, there may be isolated Read accesses while multiple processors are allowed to read the data; we can consider those as access bursts of size one. The processor starting the next burst of accesses to S-block i is chosen at random in the model. The probability that at least a Write occurs in an access burst is W and the probability that only Reads occur is $1 - W$. An access burst in which a Write occurs is called a Write access burst.

In the access burst model, the outcome is different when a Write access burst starts with a Write or with a Read [9]. Therefore, we have to define the parameter, f, the fraction of Write access bursts such that the Write occurs first; $(1 - f)$ is the fraction of access bursts such that there is at least a Read before the first Write.

3 Cache Coherence Protocols

In this section, we describe five different cache coherence protocols, analyze them, and derive closed-form formula for each coherence event based on the access burst program

model.

3.1 The Basic Coherence Protocol

This coherence protocol was described in detail in [8,9]. A block may exist in one of three states in a cache, INVALID (no copy of the block in the cache), RO (Read-Only; an arbitrary number of caches can have this block, and all the copies are identical), and RW (Read-Write; the block has been locally modified since it was brought into the cache and the main memory copy is stale).

3.1.1 Protocol Description

The Basic coherence works in steady state as follows:

1. *Read hit*: The block may be accessed locally without delay.
2. *Read miss*: If a remote cache has an RW copy of the block, the modified block must first be written back to shared memory, and then shared memory supplies the block to the requesting cache. Otherwise, the block comes directly from shared memory. Each cache with a copy of the block sets the state of its copy to RO.
3. *Write hit*: If the copy of the block is in state RO an invalidation signal must be sent to all other caches. The state of the local copy is changed to RW.
4. *Write miss*: A Write miss is treated like a *Read miss* with the following difference. If copies existed in other caches, they are invalidated and the state of the local copy is set to RW.

We can denote the state of a block in the system by 1_RW, 2_RO, ..., J_RO, where 1_RW means that the block is owned by one cache and is an RW copy; k_RO means that there are RO copies of the block in k caches. In steady-state the state 1_RO cannot be reached. State transitions occur at the end of each access burst; the protocol can be modeled as a discrete Markov chain illustrated in Figure 2.

3.1.2 Coherence Analysis

Four possible cache coherence events can occur:

1. **Miss**: this event, denoted M, occurs when a block is referenced and is not present in the cache. When a miss occurs, the block always comes from shared memory. When there are k copies of the block in k caches, a miss occurs at the beginning of a new access burst only if the processor starting a new access burst is one of the $(J - k)$ processors without a copy of the block in their caches. Hence, $P(M)$[1] is equal to

[1] In this paper, we denote by P_{state} the stationary probability of a given state in the Markov chain and by $P(event)$ the fraction of accesses causing a given event.

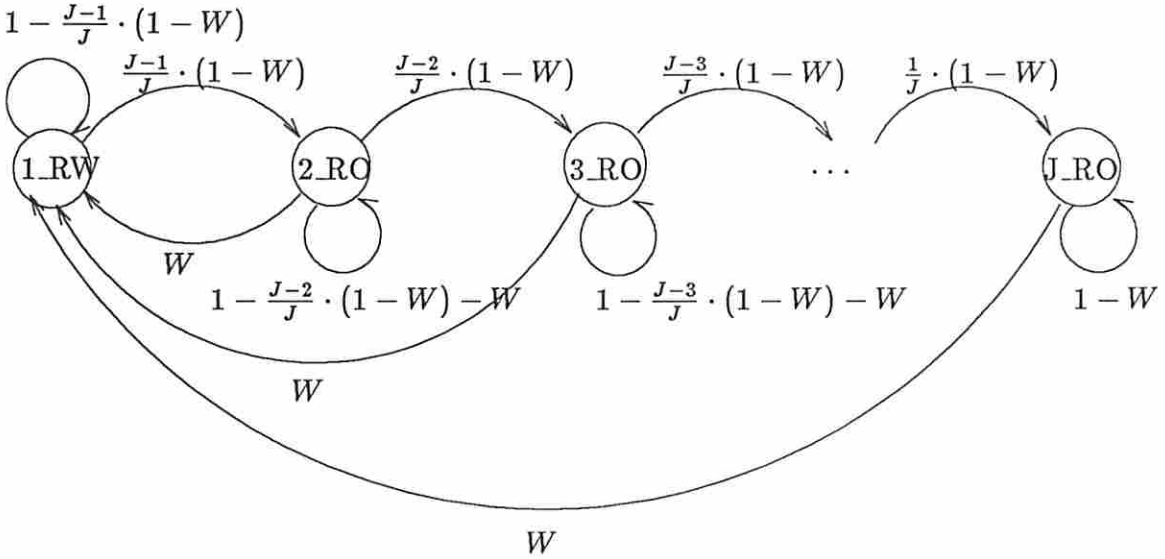


Figure 2: Markov chain for the Basic coherence protocol

$[(P_{1\_RW} \cdot (J-1)/J) + \sum_{j=2}^{J-1} P_{j\_RO} \cdot (J-j)/J]/ls$. All misses occurring as a result of the following events are accounted for as M events.

2. **Transition from RO to RW**: this event occurs when a processor needs to modify a block already present in another cache as RO. We denote this event as IN_RO (INvalidation of RO copy(ies)). An invalidation of RO copies occurs whenever an access burst modifies the block in an RO state. It also occurs in a transition from 1_RW to 1_RW, provided the second access burst is executed by a different processor and starts with a Read [9]. Therefore, $P(IN\_RO)$ is equal to $[\sum_{j=2}^{J} W \cdot P_{j\_RO} + W \cdot (1-f) \cdot P_{1\_RW} \cdot (J-1)/J]/ls$.

3. **Transition from RW to RO**: this event occurs whenever a burst leaving a block in state 1_RW is followed by a burst starting with a Read access by a different processor. This RW copy has to be written back to shared memory before shared memory supplies the block to the requesting cache. We denote this event as CS_RW (Change State of a RW copy). Therefore, $P(CS\_RW)$ is equal to $[P_{1\_RW} \cdot (1-W) \cdot (J-1)/J + P_{1\_RW} \cdot W \cdot (1-f) \cdot (J-1)/J]/ls$;

4. **Transition from RW to RW** in a different cache: this event occurs when an access burst leaving a block in state 1_RW is followed by a Write from any other processor. Like the CS_RW event, this RW copy has to be written back to shared memory before shared memory supplies the block to the requesting cache. Thus, $P(IN\_RW)$ is equal to $[W \cdot f \cdot P_{1\_RW} \cdot (J-1)/J]/ls$.

To each of these events corresponds an average penalty, λ. The penalty associated with an event is defined as the average time that a processor is blocked at each occurrence of

the event. Let's define t_{mc} and t_{inv} as the times taken by the transfer of a cache block between shared memory and a cache and by the invalidation of a block in a different cache, respectively. Thus, λ_M is equal to t_{mc}, λ_{IN_RO} is equal to t_{inv}, λ_{CS_RW} is equal to t_{mc}, and λ_{IN_RW} is equal to t_{mc}.

3.1.3 Closed form Derivation

From Figure 2, the state probability equations can be written as follows:

$$\frac{J-1}{J} \cdot (1-W) \cdot P_{1\_RW} = \sum_{j=2}^{J} W \cdot P_{j\_RO}, \tag{1}$$

$$(W + \frac{J-2}{J} \cdot (1-W)) \cdot P_{2\_RO} = \frac{J-1}{J} \cdot (1-W) \cdot P_{1\_RW}, \tag{2}$$

and

$$(W + \frac{J-k}{J} \cdot (1-W)) \cdot P_{k\_RO} = \frac{J-(k-1)}{J} \cdot (1-W) \cdot P_{(k-1)\_RO}, \quad for\ 3 \leq k \leq J. \tag{3}$$

After some transformations (See Appendix A), we have

$$P_{1\_RW} = \frac{J \cdot W}{J-1+W},$$

and

$$P(M) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W}{1+(J-1) \cdot W}.$$

Hence, the probability of each event can be written as follows:

$$P(IN\_RO) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W \cdot (1-W \cdot f)}{J-1+W}.$$

$$P(CS\_RW) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W \cdot (1-W \cdot f)}{J-1+W}.$$

$$P(IN\_RW) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W^2 \cdot f}{J-1+W}.$$

The total penalty is

$$\begin{aligned} \lambda_{total} &= P(M) \cdot \lambda_M + P(IN\_RO) \cdot \lambda_{IN\_RO} \\ &\quad + P(CS\_RW) \cdot \lambda_{CS\_RW} + P(IN\_RW) \cdot \lambda_{IN\_RW} \\ &= \frac{1}{ls} \cdot [\frac{J \cdot (J-1) \cdot W \cdot (1+W)}{(1+(J-1) \cdot W) \cdot (J-1+W)} \cdot t_{mc} + \frac{(J-1) \cdot W \cdot (1-W \cdot f)}{J-1+W} \cdot t_{inv}]. \end{aligned}$$

3.2 The Write-Once Coherence Protocol

In the Write-Once protocol [12,17], a block in a cache can be in one of four states: INVALID, VALID (as RO in the Basic protocol), RESERVED (data in the block has been locally modified exactly once since it was brought into the cache and shared memory is updated), and DIRTY (data in the block has been locally modified more than once since it was brought into the cache and the shared memory copy is stale).

3.2.1 Protocol Description

The Write-Once coherence works in steady state as follows:

1. *Read hit*: The block may be accessed locally without delay.

2. *Read miss*: If a remote cache has the copy of the block in state DIRTY, the remote cache supplies the block to the requester and updates shared memory at the same time. Otherwise, the block is loaded from shared memory. All caches having a copy of the block set its state to VALID.

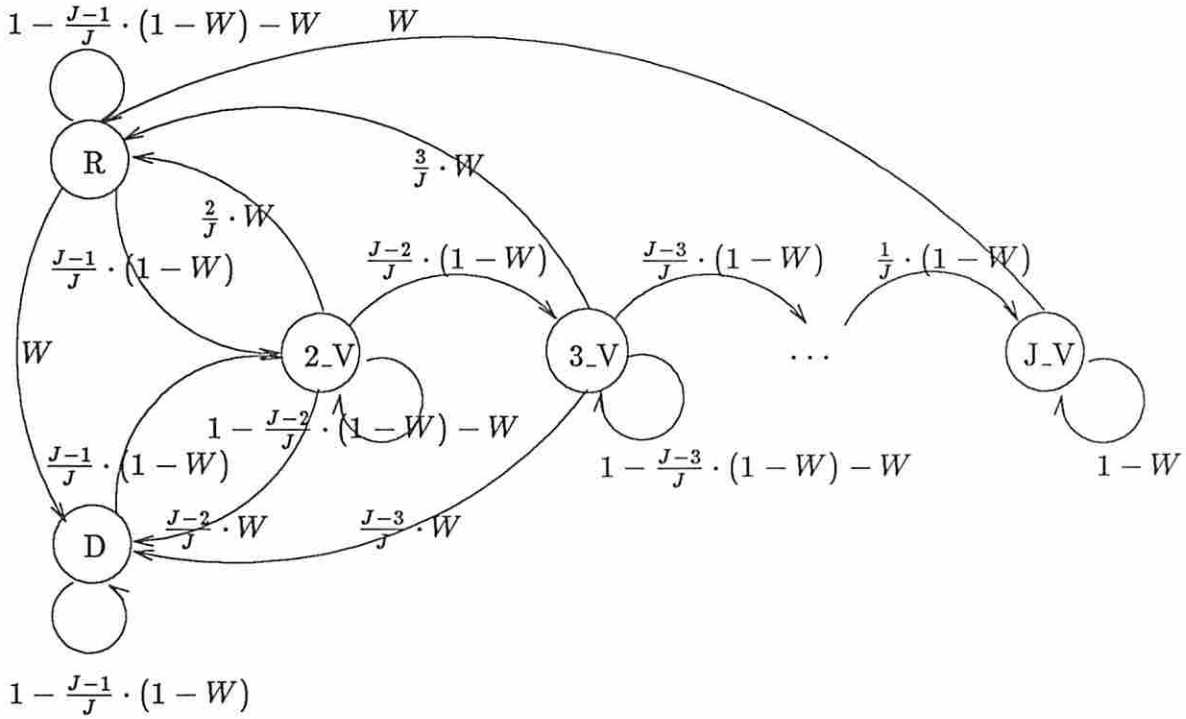
3. *Write hit*: If the block is already in state DIRTY or RESERVED, the Write can be processed locally without delay and the state of the block is always set to DIRTY. If the block is in state VALID, the word being modified is written through to shared memory, block copies in other caches are invalidated and the state of the block is set to RESERVED.

4. *Write miss*: If one remote cache owns a copy of the block in state DIRTY, the block is loaded from the remote cache and the remote cache invalidates its own copy; otherwise, the block is loaded from shared memory. Upon detecting the write miss signal on the bus, all caches with the copy of the block invalidate their copies at the same time. Once the block is loaded, the Write takes place and the state of the block is always set to DIRTY.

We denote the state of a block in the system by R, D, 2_V, ..., J_V, where R and D mean that the block is owned by one cache and is a REVERSED and DIRTY copy, respectively; k_V means that there are VALID copies of the block in k caches. The discrete Markov chain for the Write-Once coherence protocol is shown in Figure 3. This discrete Markov diagram is very similar to the one shown in Figure 2 with the following differences. The state 1_RW in Figure 2 is split into two states, R and D; at the end of each Write burst, the next state is D if a miss occurs; otherwise, the next state is R.

3.2.2 Coherence Analysis

Three possible cache coherence events can occur:



D : Dirty copy
R : Reserved copy
k_V : k processors own a valid copy

Figure 3: Markov chain for the Write-Once coherence protocol

1. **Miss:** This event is very similar as the M event in the Basic cache coherence protocol except that the block is supplied by a remote cache rather than memory if the remote cache has a DIRTY copy of the block. Hence, some misses cause cache-to-cache transfers (these miss events are denoted M_cc), and some misses cause memory-to-cache transfers (these misses are denoted M_mc.) $P(M\_cc)$ is equal to $[P_D \cdot (J-1)/J]/ls$; $P(M\_mc)$ is equal to $[(P_R \cdot (J-1)/J) + \sum_{j=2}^{J-1} P_{j\_V} \cdot (J-j)/J]/ls$;

2. **Transition from VALID to RESERVED**: this event, denoted CS_V_R (Change State from Valid to Reserved), either occurs at the end of a Write burst and no miss event happened in the burst or occurs in a transition from R to R, provided the second access burst is executed by a different processor and starts with a Read. The modified *word* is written through to shared memory. Thus, $P(CS\_V\_R)$ is equal to $[W \cdot (1-f) \cdot P_R \cdot (J-1)/J + \sum_{j=2}^{J} W \cdot P_{j\_V} \cdot j/J]/ls$.

3. **Transition from DIRTY to VALID**: this event, denoted CS_D (Change State of a DIRTY copy), is very similar to the CS_RW event except that the cache having the DIRTY copy of the block supplies the block to the requesting cache and also updates

shared memory at the same time. $P(CS\_D)$ is equal to $[P_D \cdot (1-W) \cdot (J-1)/J + P_D \cdot W \cdot (1-f) \cdot (J-1)/J]/ls$. Usually, the latency of updating shared memory is longer than that of the cache-to-cache transfer. Therefore the extra penalty to perform the miss due to the memory update is added for all events CS_D.

In addition to the t_{mc} and t_{inv} defined previously, we define two new terms, t_{word} and t_{cc}, which are the times to write a word to shared memory and to transfer a block between two caches, respectively. Hence, λ_{M_mc} is equal to t_{mc}. λ_{M_cc} is equal to t_{cc}. $\lambda_{IN\_V\_R}$ which is equal to t_{word}. λ_{CS_D} is equal to $(t_{mc} - t_{cc})$.

3.2.3 Closed form Derivation

From Figure 3, We may equate the state probability equations as follows:

$$\frac{J-1}{J} \cdot (1-W) \cdot P_D = W \cdot P_R + \sum_{j=2}^{J} \frac{J-j}{J} \cdot W \cdot P_{j\_V},$$

$$(W + \frac{J-1}{J} \cdot (1-W)) \cdot P_R = \sum_{j=2}^{J} \frac{j}{J} \cdot W \cdot P_{j\_V},$$

$$(W + \frac{J-2}{J} \cdot (1-W)) \cdot P_{2\_V} = \frac{J-1}{J} \cdot (1-W) \cdot (P_D + P_R),$$

and

$$(W + \frac{J-k}{J} \cdot (1-W)) \cdot P_{k\_V} = \frac{J-(k-1)}{J} \cdot (1-W), \cdot P_{(k-1)\_V}, \quad for\ 3 \leq k \leq J.$$

After some similar transformations to those in Appendix A, we have

$$P_D = \frac{J \cdot W^2 \cdot (J^2 + 2JW - 2J - 2W + 2)}{(J-1+W)^2 \cdot (1+(J-1)W)},$$

$$P_R = \frac{J \cdot W \cdot (J - J \cdot W^2 + W^2 - 1)}{(J-1+W)^2 \cdot (1+(J-1)W)},$$

Hence, the probability of each event can be written as follows:

$$P(M\_cc) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W^2 \cdot (J^2 + 2JW - 2J - 2W + 2)}{(J-1+W)^2 \cdot (1+(J-1)W)}],$$

$$P(M\_mm) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (1-W) \cdot (J^2 + 2JW - 2J - 3W + 1)}{(J-1+W)^2 \cdot (1+(J-1)W)}].$$

$$P(CS\_V\_R) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (JW^2 - 2W^2 + W + 1)}{(J-1+W) \cdot (1+(J-1)W)} - \frac{(J-1) \cdot W^2 \cdot f}{J-1+W}],$$

and

$$P(CS\_D) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W^2 \cdot (1-fW) \cdot (J^2 + 2JW - 2J - 2W + 2)}{(J-1+W)^2 \cdot (1+(J-1)W)}].$$

The total penalty is:

$$\begin{aligned}
\lambda_{total} &= P(M_cc) \cdot \lambda_{M_cc} + P(M_mc) \cdot \lambda_{M_mc} \\
&\quad + P(IN_V_R) \cdot \lambda_{IN_V_R} + P(CS_D) \cdot \lambda_{CS_D} \\
&= \frac{1}{ls} \cdot [\frac{(J-1) \cdot W}{(1+(J-1) \cdot W)} \cdot t_{mc} - \frac{(J-1) \cdot W^2 \cdot f}{J-1+W} \cdot t_{word} \\
&\quad + \frac{(J-1) \cdot W^3 \cdot f \cdot (J^2 + 2JW - 2J - 2W + 2)}{(J-1+W)^2 \cdot (1+(J-1)W)} \cdot t_{cc} \\
&\quad - \frac{(J-1) \cdot W^3 \cdot f \cdot (J^2 + 2JW - 2J - 2W + 2)}{(J-1+W)^2 \cdot (1+(J-1)W)} \cdot t_{mc} \\
&\quad + \frac{(J-1) \cdot W \cdot (JW^2 - 2W^2 + W + 1)}{(J-1+W) \cdot (1+(J-1)W)} \cdot t_{word}].
\end{aligned}$$

3.3 The Synapse Coherence Protocol

In the Synapse protocol [11], there is a single bit tag with each cache block in shared memory, indicating whether shared memory is to respond to a miss on that block. If a remote cache has a modified copy of the block, the bit will inhibit shared memory from supplying the block. Hence, this bit can prevent a possible race condition when the remote cache does not respond quickly enough to inhibit shared memory.

A cache block may be in one of the three states: INVALID, VALID (as RO in the Basic protocol), and DIRTY (as RW in the Basic protocol).

3.3.1 Protocol Description

The Synapse coherence protocol works in steady state as follows:

1. *Read hit*: The access may be processed locally without delay.

2. *Read miss*: If a remote cache has a DIRTY copy of the block, the modified block must first be written back to shared memory; the tag bit of the block in shared memory is set; the remote cache invalidates its local copy and sends a busy acknowledge signal to the requesting cache. When the requesting cache receives this busy signal, it must

send an additional read miss request in order to get the copy of the block from shared memory. In all other cases the block is directly supplied by the shared memory. The state of the loaded block is always set to VALID.

3. *Write hit*: If the block is in state DIRTY in local cache, the Write can be processed locally without delay. If the local copy of the block is in state VALID, the procedure is as follows: shared memory has to transfer the ownership along with the copy to the requesting cache and each cache with a copy of the block observes this bus transaction and invalidates its copy of the block at the same time.

4. *Write miss*: If a remote cache has a DIRTY copy of the block, the remote cache transfers the ownership along with the block copy to the requester. If all copies of the block in the system are VALID, shared memory supplies the copy to the requesting cache and each cache which has a VALID block copy invalidates its copy at the same time. The tag bit in shared memory is reset.

We can denote the states of a block in the system by D, 1_V, 2_V, ..., J_V, where D is the state in which the block is owned by one cache and is a DIRTY copy; k_V means that there are VALID copies of the block in k caches. If we observe state transitions at the end of each access burst, then the discrete Markov chain of the Synapse coherence protocol can be drawn and is shown in Figure 3. This discrete Markov diagram is very similar as the one shown in Figure 2 except that one more state, 1_V, is introduced.

3.3.2 Coherence Analysis

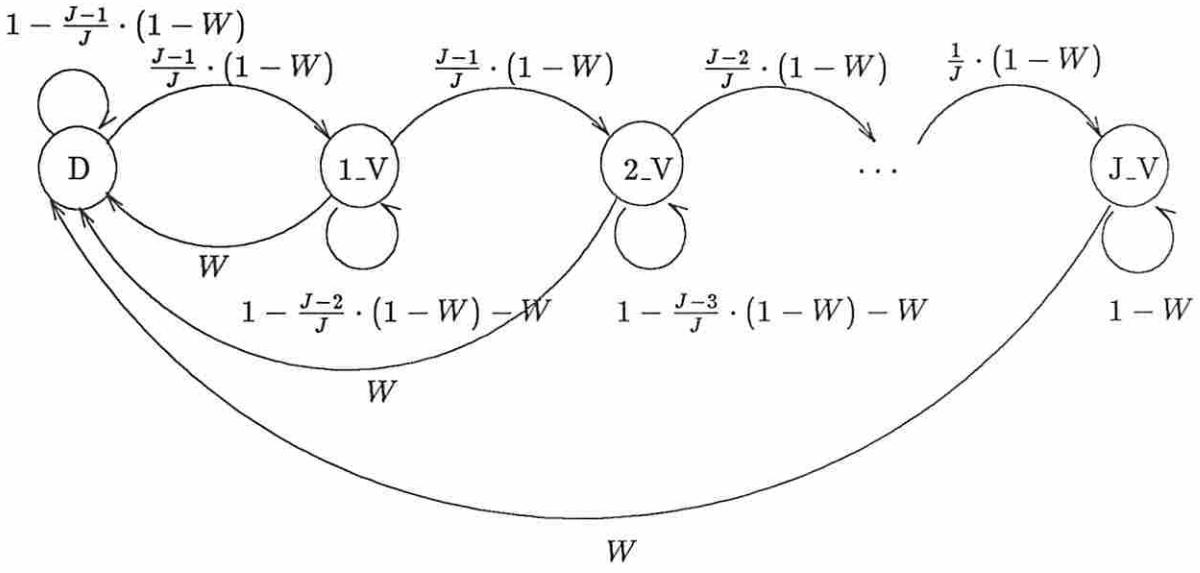
Three possible cache coherence events can occur:

1. **Miss**: There are two types of miss events (as in the Write-Once protocol); these events are denoted M_cc and M_mc for the cases of a cache-to-cache and memory-to-cache transfers respectively. A miss causes a cache-to-cache transfer when a remote cache has a DIRTY copy of the block and the access burst is a Write burst. Therefore, $P(M\_cc)$ is equal to $[W \cdot P_D \cdot (J-1)/J]/ls$; $P(M\_mc)$ is equal to $[((1-W) \cdot P_D \cdot (J-1)/J) + \sum_{j=1}^{J-1} P_j \cdot (J-j)/J]/ls$;

2. **Transition from VALID to DIRTY on hit**: this event, denoted IN_V_h (IN-validation of Valid Copy(ies) on hit), either occurs at the end of a Write burst and no miss event happened in the burst or occurs in a transition from D to D, provided the second access burst is executed by a different processor and starts with a Read. This event includes a block transfer from shared memory to the requesting cache. Thus, $P(IN\_V\_h)$ is equal to $[W \cdot (1-f) \cdot P(D) \cdot (J-1)/J + \sum_{j=1}^{J} W \cdot P_{j\_V} \cdot j/J]/ls$;

3. **Transition from DIRTY to VALID**: this event, denoted CS_D, are the same as the CS_RW event in the Basic cache coherence protocol. $P(CS\_D)$ is equal to $[P_D \cdot (1-W) \cdot (J-1)/J + P_D \cdot W \cdot (1-f) \cdot (J-1)/J]/ls$.

The penalty of each event is thus the follows: λ_{M_cc} is equal to t_{cc}; λ_{M_mc} is equal to t_{mc}; $\lambda_{IN\_V\_h}$ is equal to t_{mc}; and λ_{CS_D} is equal to t_{mc}.



i_V : i processors own the valid copy

D : Dirty Copy

Figure 4: Markov chain for the Synapse coherence protocol

3.3.3 Closed form Derivation

From Figure 4, we may derive the state probabilities from the following equations:

$$\frac{J-1}{J}\cdot(1-W)\cdot P_D=\sum_{j=1}^{J}W\cdot P_{j\_V},$$

$$(W+\frac{J-1}{J}\cdot(1-W))\cdot P_{1\_V}=\frac{J-1}{J}\cdot(1-W)\cdot P_D,$$

and

$$(W+\frac{J-k}{J}\cdot(1-W))\cdot P_{k\_V}=\frac{J-(k-1)}{J}\cdot(1-W)\cdot P_{(k-1)\_V},\quad for\ 2\le k\le J.$$

After some similar transformations to those in Appendix A, we have

$$P_D=\frac{J\cdot W}{J-1+W},$$

$$P(M\_cc) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W^2}{J-1+W}],$$

$$P(M\_mc) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (1-W) \cdot (J+JW-W)}{(J-1+W) \cdot (1+(J-1)W)}].$$

The probability of other events can be written as follows:

$$P(IN\_V\_h) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (1+JW^2-W^2-fW \cdot (1+(J-1)W))}{(J-1+W) \cdot (1+(J-1)W)}],$$

$$P(CS\_D) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (1-fW)}{J-1+W}].$$

The total penalty for the Synapse protocol is:

$$\begin{aligned}
\lambda_{total} &= P(M_cc) \cdot \lambda_{M_cc} + P(M_mc) \cdot \lambda_{M_mc} \\
&\quad + P(IN_V_h) \cdot \lambda_{IN_V_h} + P(CS_D) \cdot \lambda_{CS_D} \\
&= \frac{1}{ls}[\frac{(J-1) \cdot W^2}{J-1+W} \cdot t_{cc} + \frac{(J-1) \cdot W \cdot (JW-2W+J+2)}{(J-1+W) \cdot (1+(J-1)W)} \cdot t_{mc} \\
&\quad - \frac{2 \cdot (J-1) \cdot W^2 \cdot f}{J-1+W} \cdot t_{mc}].
\end{aligned}$$

3.4 The Illinois Coherence Protocol

In the Illinois protocol [15], a block in a cache can be in one of the four states: INVALID, EXCL-UNMOD (Exclusive-Unmodified; no other cache has this block; data in block is consistent with shared memory), SHARED-UNMOD (Shared-Unmodified; as RO in the Basic protocol) and EXCL-MOD (Exclusive-Modified; as RW in the Basic protocol).

3.4.1 Protocol Description

The scheme works in steady state as follows:

1. *Read hit*: The access may be processed locally without delay.

2. *Read miss*: If a remote cache has an EXCL-MOD copy of the block, the remote cache sends the copy to the requesting cache and updates shared memory at the same time. Otherwise, any one cache supplies the copy to the requester. Both caches set their copy to SHARED-UNMOD.

3. *Write hit*: If the local copy of the block is in state EXCL-MOD, it can be updated without delay. Otherwise, the Write cannot be processed until an invalidation signal is sent. The copy in the local cache is set to EXCL-MOD.

4. *Write miss*: A *Write miss* request is broadcasted to all caches. Each cache with the copy of the block invalidates its copy. The block is always loaded from a remote cache and its state is set to EXCL-MOD.

We can denote the state of a block in the system by E, 2_S, ..., J_S, where E means that the block is owned by one cache and is an EXCL-MOD copy; k_S means that there are SHARED-UNMOD copies of the block in k caches. The discrete Markov chain of the Illinois coherence protocol is the same as the one shown in Figure 2 provided that the state names are changed.

3.4.2 Coherence Analysis

Three possible cache coherence events can occur:

1. **Miss:** This event is very similar to the M event in the Basic cache coherence protocol except that the block is always supplied by a cache. $P(M)$ is equal to $[(P_E \cdot (J-1)/J) + \sum_{j=2}^{J-1} P_{j\_S} \cdot (J-j)/J]/ls$.

2. **Transition from SHARED-UNMOD to EXCL-MOD on hit:** This event, denoted IN_S_h (INvalidation of SHARED-UNMOD Copy(ies) on hit), and the IN_V_h event in the Synapse cache coherence protocol are very similar except that the coherence overhead of this event is to broadcast an invalidation signal. $P(IN\_S\_h)$ is equal to $[W \cdot (1-f) \cdot P_E \cdot (J-1)/J + \sum_{j=2}^{J} W \cdot P_{j\_S} \cdot j/J]/ls$.

3. **Transition from EXCL-MOD to SHARED-UNMOD:** This event, denoted CS_E (Change State of an EXCL-MOD copy), is very similar to the CS_D event in the Write-Once cache coherence protocol except that the penalty for this event is the time difference between a cache-to-memory transfer and a cache-to-cache transfer. $P(CS\_E)$ is equal to $[P_E \cdot (1-W) \cdot (J-1)/J + P_E \cdot W \cdot (1-f) \cdot (J-1)/J]/ls$.

The penalty of each event is: λ_M is equal to t_{cc}, $\lambda_{IN\_S\_h}$ is equal to t_{inv}, and λ_{CS_E} is equal to $(t_{mc} - t_{cc})$.

3.4.3 Closed form Derivation

The state probability equations of the Illinois cache coherence protocol are the same as those of the Basic cache coherence protocol. Hence, the probability of each event can be written as follows:

$$P(M) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W}{1 + (J-1) \cdot W}].$$

$$P(IN\_S\_h) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (JW^2 - 2W^2 + W + 1 - Wf - JW^2 f + W^2 f)}{(J-1+W) \cdot (1 + (J-1)W)}],$$

$$P(CS\_E) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (1 - W \cdot f)}{J - 1 + W}].$$

And the total penalty for the Illinois protocol is:

$$\begin{aligned}
\lambda_{total} &= P(M) \cdot \lambda_M + P(IN_S_h) \cdot \lambda_{IN_S_h} + P(CS_E) \cdot \lambda_{CS_E} \\
&= \frac{1}{ls} \cdot [\frac{(J-1) \cdot (J-2) \cdot W \cdot (1-W)}{(J-1+W) \cdot (1+(J-1)W)} \cdot t_{cc} + \frac{(J-1) \cdot W^2 \cdot f}{J-1+W} \cdot t_{cc} \\
&\quad + \frac{(J-1) \cdot W \cdot (1 - W \cdot f)}{J-1+W} \cdot t_{mc} + \frac{(J-1) \cdot W \cdot (1-W^2)}{(J-1+W) \cdot (1+(J-1)W)} \cdot t_{inv} \\
&\quad + \frac{(J-1) \cdot W^2 \cdot (1-f)}{J-1+W} \cdot t_{inv}].
\end{aligned}$$

3.5 The Berkeley Coherence Protocol

In the Berkeley protocol [13], a block in a cache can be in one of the following four states: INV (INValid; as INVALID in the Basic protocol), UNO (UNOwned; as RO in the Basic protocol), EXC (owned EXClusively; the block copy is unique, and therefore it can be updated locally without delay; the cache must respond to any request on the bus for a copy of the block; this state is equivalent to the RW in the Basic protocol), or NON (Owned NON-exclusively; the block copy is owned, but it cannot be modified without informing the other caches). At any time up to one NON copy and several UNO copies of a block can exist. In steady state, there is one and only one NON copy of a block in the system if there exist some UNO copies of the block; on the other hand, there is never a NON copy of the block in the system if there is an EXC copy of the block. The cache, which has a copy of the block in state NON or EXC, is called the owner of the block. If a block is not owned by any cache, shared memory is the owner; in a system with infinite caches, in which replacements never occur, the memory cannot be an owner in steady state.

3.5.1 Protocol Description

The Berkeley protocol works as follows in steady state, for the case of infinite caches.

1. *Read hit*: The access is processed locally without delay.

2. *Read miss*: The block is always loaded from another cache and its local state is set to UNO.

3. *Write hit*: If the local copy of the block is in state EXC, the Write is processed without delay. Otherwise, all copies must be invalidated before the Write can be processed; the cache sets its copy to state NON.

4. *Write miss*: The block always comes from another cache and each cache with the copy of the block invalidates its copy. The requesting cache sets its copy to state EXC.

We can denote the state of a block in the system by E, 2_N, ..., J_N, where E means that the block is owned by one cache and is an EXC copy; k_N means that there are one NON and $(k-1)$ UNO copies of the block in k caches. The discrete Markov chain of the Berkeley coherence protocol can be derived; provided the state names are changed, the Markov chain is the same as the one shown in Figure 2.

3.5.2 Coherence Analysis

In this scheme, two possible cache coherence events can occur:

1. **Miss:** The fraction of misses in this protocol is given by the same expression as in the Illinois protocol, i.e., $[P(M) = (P_E \cdot (J-1)/J) + \sum_{j=2}^{J-1} P_{j\_N} \cdot (J-j)/J]/ls$.

2. **Transition from UNO to NON on hit:** The fraction of references causing this event IN_U_h (INvalidation of UNO Copy(ies) on hit), is given by the same expression as for the event IN_S_h in the Illinois protocol, that is, $[P(IN\_U\_h) = W \cdot (1-f) \cdot P_E \cdot (J-1)/J + \sum_{j=2}^{J} W \cdot P_{j\_N} \cdot j/J]/ls$

The penalty of each event is: $\lambda_M = t_{cc}$, and $\lambda_{IN\_U\_h} = t_{inv}$.

3.5.3 Closed form Derivation

The state probability equations of the Berkeley cache coherence protocol is the same as those of the Basic cache coherence protocol. Hence, the probability of each event can be written as follows:

$$P(M) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W}{1+(J-1)W}].$$

$$P(IN\_U\_h) = \frac{1}{ls} \cdot [\frac{(J-1) \cdot W \cdot (JW^2 - 2W^2 + W + 1 - Wf - JW^2 f + W^2 f)}{(J-1+W) \cdot (1+(J-1)W)}].$$

And the total penalty is:

$$\begin{aligned} \lambda_{total} &= P(M) \cdot \lambda_M + P(IN\_U\_h) \cdot \lambda_{IN\_U\_h} \\ &= \frac{1}{ls} \cdot [\frac{(J-1) \cdot W}{1+(J-1)W} \cdot t_{cc} + \frac{(J-1) \cdot W \cdot (1-W^2)}{(J-1+W) \cdot (1+(J-1)W)} \cdot t_{inv} \\ &\quad + \frac{(J-1) \cdot W^2 \cdot (1-f)}{J-1+W} \cdot t_{inv}]. \end{aligned}$$

4 Multitasked S.O.R. Algorithm

In this section, we apply the model to one particular multitasked algorithm, the S.O.R. (Successive Over Relaxation) iterative algorithm, to solve Laplace's equation $\nabla^2 x = 0$ on a rectangular domain of R^2. The model was compared with trace-driven simulations of several algorithms in [9]. The S.O.R. algorithm is an iterative, compute-intensive algorithm. The infinite cache condition is met when the data cache of each processor is large enough to contain all the grid elements accessed by the processor. In this case, steady-state is reached after the first iteration. This important algorithm is therefore a good benchmark to apply the model. The details of the algorithm can be found in [7] and many other sources. The algorithm consists in updating the points of a rectangular grid, in each iteration.

Let $L \cdot N$ be the size of the rectangular grid; N is the horizontal dimension and L is the vertical dimension. A partition can be described in terms of P areas, where P is the number of processors. $P = qp$, where p and q are the number of areas on the horizontal and the vertical sides of the grid. We assume that P is a power of 2. If P is a square number, we choose $p = q = \sqrt{P}$; otherwise, we choose $q = \sqrt{P/2}$ and $p = P/q = \sqrt{2 \cdot P}$. This partitioning strategy is shown in Figure 5. The boundary conditions add a total of $2 \cdot (L+N)$ grid points, so that the total number of grid points is actually $L \cdot N + 2 \cdot (L+N)$.

Grid elements which are in even position (the sum of the indexes is even) are tagged as red and grid elements in odd positions are tagged as black. Red elements are updated in the first sweep of an iteration and black elements are then updated in the second sweep of the same iteration. In a multiprocessor system, a natural decomposition of this problem is to allocate one partition of the grid points to each processor. Each processor has the same number of red and black iterates. After each sweep, the processors have to synchronize. In general, a processor has to synchronize with at most 4 neighbors [9].

In the algorithm, the equation for the update of an iterate in the $(K+1)$th iteration is:

$$x_{i,j}^{(K+1)} = (1-w) \cdot x_{i,j}^{(K)} + \frac{1}{4} \cdot w \cdot [x_{i+1,j}^{(K)} + x_{i-1,j}^{(K)} + x_{i,j+1}^{(K)} + x_{i,j-1}^{(K)}]$$

where w is the relaxation factor.

Figure 5 illustrates the case of a grid of size 16 X 16 and of four processors. In the following, we consider the case of a grid size 128 X 128 and four data elements in a cache block. When a cache block contains four data elements, eight different types of S-blocks can be distinguished (see Figure 5), called type 1 to type 8. All the blocks with the same type are accessed with the same pattern, and form a set in the sense defined in Section 2. There are 124 type 1 S-blocks; each type 1 S-block is referenced 24 times in an iteration (five Reads and one Write per data element); the total number of references in an iteration are 98304 ($128 \cdot 128 \cdot 6$); thus, the value of q_s for type 1 S-blocks is 0.003027. Type 1 S-blocks are shared by two processors, that is, $J = 2$. The reference pattern of a type 1 S-block in an iteration has four Write bursts (a write burst occurs when an element of the

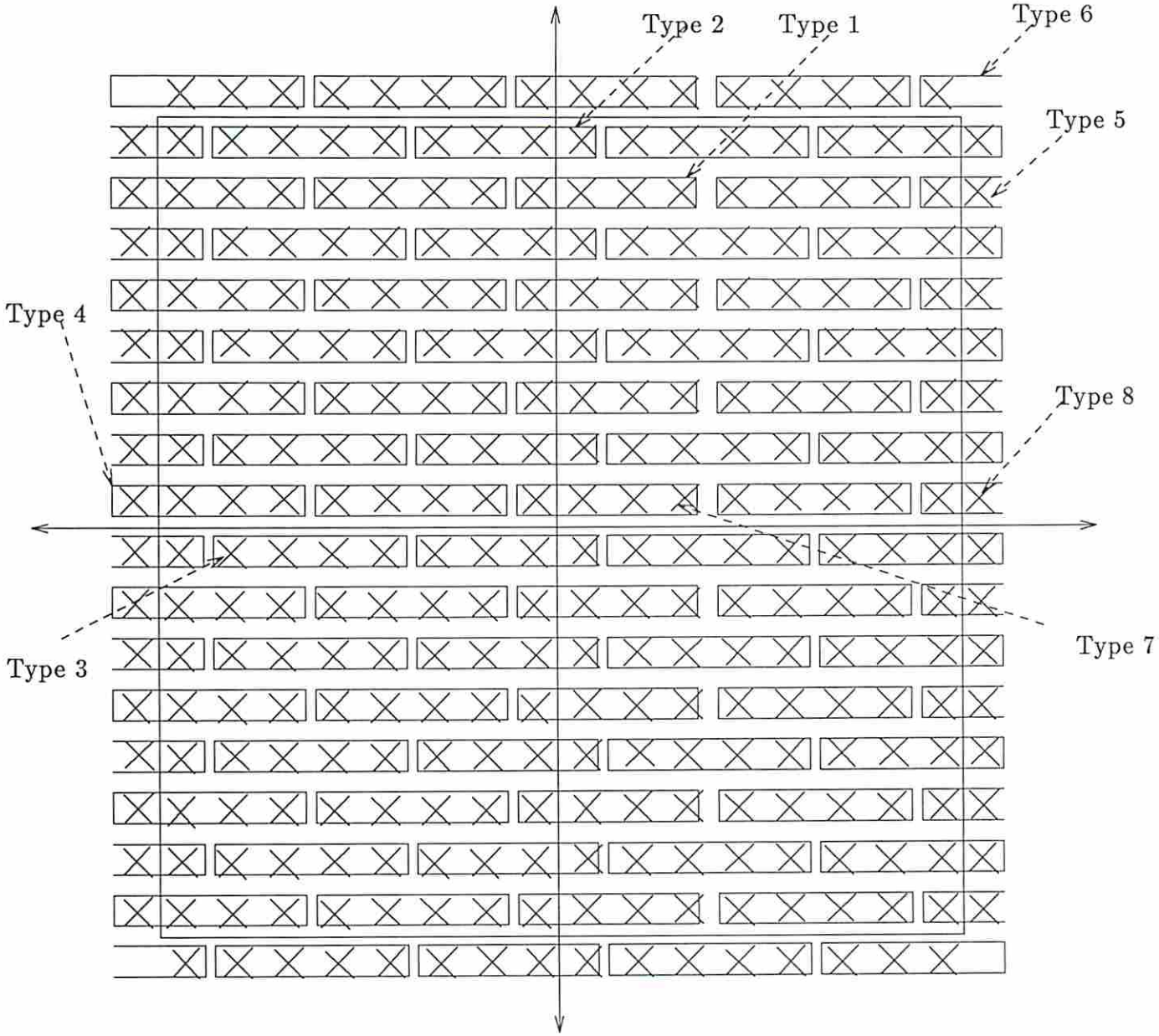


Figure 5: The eight sets of S-blocks in the S.O.R. iterative when a cache block contains four data elements, the grid size is 16 X 16, and the number of processors is four.

block is updated in a sweep); the total number of accesses to the block for the update of the four elements is 14; there are 10 isolated Read besides the Write bursts; thus the value of W is $4/(10+4) = 0.2857$ and ls is $24/(10+4) = 1.7143$. The first reference in a Write burst of type 1 S-block is always a Read, that is, the value of f is zero. Similarly, we can compute the value of these parameters for type 2 to type 8 S-blocks; these values are listed in Table 1.

Table 1: Values of parameters for the eight different sets of S-blocks in the case of the S.O.R. iterative algorithm with a grid size of 128 X 128.

Set	q_s	J	W	ls	f
Type 1	0.03027	2.0000	0.2857	1.7143	0.0000
Type 2	0.00041	2.0000	0.4000	2.0000	0.0000
Type 3	0.01465	2.0000	0.1667	2.0000	0.0000
Type 4	0.00037	2.0000	0.2222	2.0000	0.0000
Type 5	0.00757	2.0000	0.2500	1.5000	0.0000
Type 6	0.00012	2.0000	0.2500	1.5000	0.0000
Type 7	0.00049	4.0000	0.2857	1.7143	0.0000
Type 8	0.00012	4.0000	0.2500	1.5000	0.0000

In the computation of the total penalty, the cache-to-cache transfer time is taken as eight time units: one time unit for bus arbitration, one time unit for address transfer, four time units for a block access and transfer, and two time units for acknowledgement. The memory-to-cache transfer time is taken as ten time units because an access to the memory takes six time units. The time to write a word to shared memory is seven time units: one time unit for bus arbitration, one time unit for address transfer, three time units for a word transfer and memory access, and two time units for acknowledgement. An invalidation signal only takes two time units: one for bus arbitration and one for signal broadcasting.

In the following, we will express all penalties in units of the penalty of transferring a single word between a cache and the shared memory, that is, $\lambda_{word} = 1$. If the penalty to read a word from memory is the same as the penalty to write a word to memory, then we can estimate the performance improvement due to the caching of shared writable data as $1 - \lambda_{total}$. In particular if $\lambda_{total} > 1$, then caching shared writable data is not productive. Thus, the penalties of different coherence events are $t_{mc} = 10/7$, $t_{cc} = 8/7$, $t_{word} = 1$ and $t_{inv} = 2/7$.

From Table 1, we can calculate the miss ratio, M, and the total penalty, λ_{total}, for the S.O.R. iterative algorithm for the five different protocols; the results are compared to the results of trace-driven simulations for the five protocols in Table 2 and Table 3[2]; the difference between model predictions and simulations is never more than 5%.

Table 2: Miss ratio, M, comparison between model prediction and simulations

Protocol	Model Prediction	Simulation Result	Difference (%)
Basic	0.006254	0.006559	4.65%
Write-Once	0.006254	0.006559	4.65%
Synapse	0.009880	0.009766	1.17%
Illinois	0.006254	0.006559	4.65%
Berkeley	0.006254	0.006559	4.65%

Table 3: Total penalty, λ_{total}, comparison between model prediction and simulations

Protocol	Model Prediction	Simulation Result	Difference (%)
Basic	0.01953	0.02047	4.59%
Write-Once	0.01510	0.01583	4.61%
Synapse	0.02996	0.03058	2.03%
Illinois	0.01068	0.01119	4.56%
Berkeley	0.00891	0.00934	4.62%

5 Discussion

The performance indexes which can be derived from the model are the *miss ratio* and the *total coherence penalty*. This discussion applies for steady-state computations in systems with infinite caches, and for data sharing patterns for which the access burst model is acceptable.

[2]Trace-driven simulation has a drawback that the same trace is re-used to evaluate all coherence protocols, while in reality the reference pattern is different for each coherence scheme because of the timing differences. Nevertheless, the traces represent at least one possible run of a real program which can accurately distinguish the performance of various coherence protocols for that run. [1]

5.1 Miss Ratio

The miss ratio is given by the sum of $P(M\_cc)$ and $P(M\_mc)$. The Basic, the Write-once, the Illinois and the Berkeley coherence protocols have very similar Markov chains, and the same miss ratio, which is

$$P(M) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W}{1 + (J-1) \cdot W} \tag{4}$$

The transition from state 1_RW to state 2_RO is replaced by a transition from D to 1_V in the Synapse coherence protocol; in this case the miss ratio is

$$P(M) = \frac{1}{ls} \cdot \frac{J \cdot (J-1) \cdot W}{(J-1+W) \cdot (1 + (J-1) \cdot W)} \tag{5}$$

This value is always higher than the value of equation (4) since J is always larger than $J-1+W$ ($W \leq 1$). The miss ratio is displayed in Figure 6 for different value of J and for W=0.25 and in Figure 7 for different value of W and for J=16. The access burst model predicts that the miss ratio on S-block i shared by 16 processors is more than $0.6/l_s$, even for cases where W is less than 15%. Also, when either W=0.25 and $J \geq 16$ or J=16 and $W \geq 0.25$, miss ratio is insensitive to either J or W; however, if we can double the value of ls, the miss ratio will be halved.

5.2 Total Penalty

Figures 8 and 9 display the product (penalty $\times$ l_s) as a function of W and J when f=1. Figures 10 and 11 show the product (penalty $\times$ l_s) as a function of W and J when f=0 (two extreme cases). From these 4 figures, the Berkeley coherence protocol always shows the best performance. The Illinois coherence protocol always has less total penalty than the Write-once coherence protocol. The Basic or the Synapse coherence protocols always exhibit the worst performance. Our examples show that, under the access burst model with f=1, with J less than 10 and W=0.25, the Synapse coherence protocol shows the worst performance; however, when J is larger than 10 and W=0.25, the Basic coherence protocol has the worst penalty for shared data accesses; when f=0, the total penalty of the Synapse protocol is always higher than that of the Basic protocol. These conclusions are similar to Archibald and Baer's, in [3]. However the conclusion may vary for different values of the penalties, which in turn depend on the architecture of the system. The model permits rapid evaluations of the protocols for given parameter values.

6 Conclusion

In this paper, we have applied a program model to five coherence protocols for the caching of shared writable data in multiprocessor systems with infinite caches. The trend towards

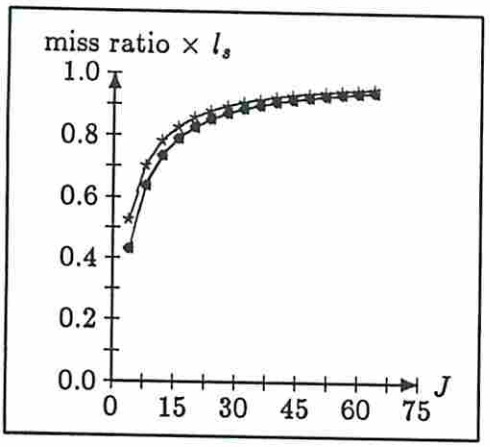


Figure 6: miss ratio × l_s for access burst model (W=0.25)

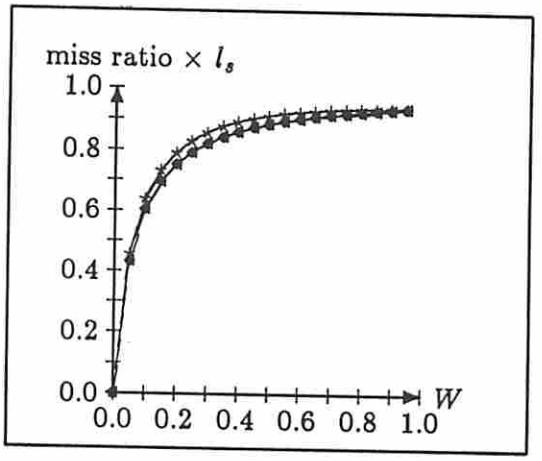


Figure 7: miss ratio × l_s for access burst model (J=16)

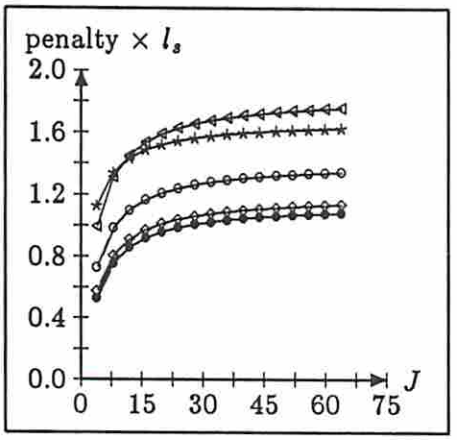


Figure 8: penalty × l_s for access burst model (W=0.25, f=1)

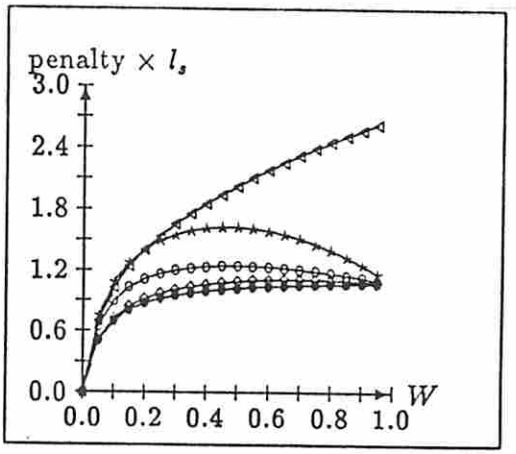


Figure 9: penalty × l_s for access burst model (J=16, f=1)

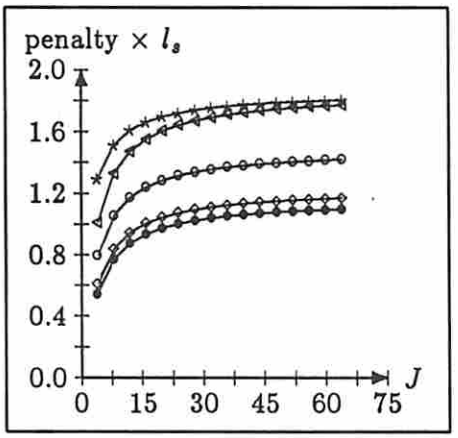


Figure 10: penalty × l_s for access burst model (W=0.25 and f=0)

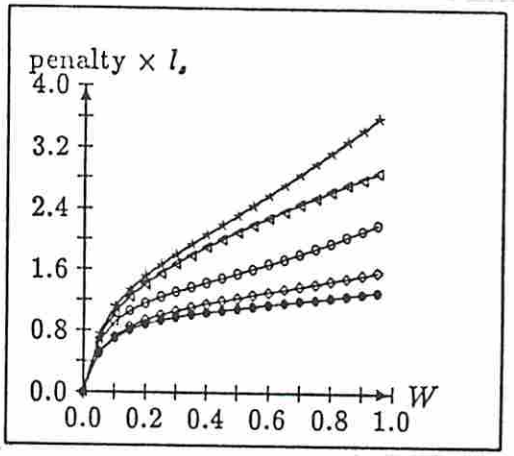


Figure 11: penalty × l_s for access burst model (J=16 and f=0)

◁ : Basic ○ : Write-Once ✶ : Synapse ◇ : Illinois ● : Berkeley

large caches seems inevitable in general-purpose computing because of the large hit ratio required by more powerful processors and because of the expected availability in a few years of VLSI chips with several megabytes of memory. Iterative, compute-intensive algorithms occur very often in multiprocessor algorithms [4]. The hypothesis of infinite caches is obtained when the data cache is large enough to contain all the data accessed by each processor, and steady state is reached after the first iteration.

The infinite cache hypothesis removes one of the major problem of analytical program models, namely the fact that such models were not good at predicting hit rates. Our model does not capture the locality of each data stream, which should include the private data as well, but only the sharing of blocks. It is also clear that the model will not apply to all possible algorithms. In fact, in [9] and [10], an example is given for which the model breaks down. This occurs when the pattern of sharing among processors does not fit the model. For instance, if a block is accessed successively by J different processors, but after it has accessed it a processor never accesses it again, then the model gives poor predictions. The model however works well when the sharing occurs back and forth among a set of processors. The reason is that we assume that the next processor to access the block after a burst is drawn at random among the J processors.

It is remarkable that for the access burst model for accessing shared data and under the infinite cache assumption all coherence components are very simple. The maximum number of parameters needed for the most complicated case is no more than four. The infinite cache results for a given set of S-blocks are independent of all cache parameters (organization, replacement policy) and of the reference pattern to other blocks. This is a considerable advantage, if one considers the complexity of the problem investigated in this paper.

References

[1] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. An evaluation of directory schemes for cache coherence. In *Proceedings of 15th Annual International Symposium on Computer Architecture*, pages 280–289, June 1988.

[2] G.R. Andrews and F.B. Schneider. Concepts and notations for concurrent programming. *Computing Surveys*, 15(1), March 1983.

[3] J. Archibald and J.L. Baer. Cache-coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4(4):273–298, November 1986.

[4] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation.* Prentice-Hall, 1989.

[5] IBM Corporation. Special issue on IBM 3081. *IBM Journal of RES. and Devel.*, 26(1):2–19, January 1982.

[6] P.J. Courtois, F. Heymans, and D.L. Parnas. Concurrent control with readers and writers. *Communications of the ACM*, 14(10):667–668, October 1971.

[7] M. Dubois. Throughput analysis of cache-based multiprocessors with multiple buses. *IEEE Transactions on Computers*, C-37(1):58–70, January 1988.

[8] M. Dubois and F.A. Briggs. Effects of cache coherency in multiprocessors. *IEEE Transactions on Computers*, C-31(11):1083–1099, November 1982.

[9] M. Dubois and J.C. Wang. Shared data contention in a cache coherence protocol. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 146–155, August 1988.

[10] S.J. Eggers and R.H. Katz. A characterization of sharing in parallel programs and its application to coherency protocol evaluation. In *Proceedings of 15th Annual International Symposium on Computer Architecture*, pages 373–382, June 1988.

[11] S.J. Frank. Tightly coupled multiprocessor system speeds memory-access times. *Electronics*, 57(1):164–169, January 1984.

[12] J.R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of 10th Annual International Symposium on Computer Architecture*, pages 124–131, June 1983.

[13] R.H. Katz, S.J. Eggers, D.A. Wood., C.L. Perkins and R.G. Sheldon. Implementing a cache consistency protocol. In *Proceedings of 12th Annual International Symposium on Computer Architecture*, pages 276–283, June 1985.

[14] E. McCreight. The dragon computer system: an early overview. In *NATO Advanced Study Institute on Microarchitecture of VLSI Computers*, July 1984.

[15] M.S. Papamarcos and J.H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proceedings of 11th Annual International Symposium on Computer Architecture*, pages 348–354, June 1984.

[16] C.P. Thacker, L.C. Stewart, and E.H. Satterthwaite, Jr. Firefly: a multiprocessor workstation. *IEEE Transactions on Computers*, C-37(8):909–920, August 1988.

[17] A.W. Wilson Jr. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proceedings of 14th Annual International Symposium on Computer Architecture*, pages 244–252, June 1987.

Appendix A

Let's denote

$$X_1 = P_{1\_RW} + \sum_{k=2}^{J} k \cdot P_{k\_RO}$$

and

$$X_2 = P_{1\_RW} + \sum_{k=2}^{J} k^2 \cdot P_{k\_RO}$$

After multiply both sides of equation (refeqn:m2) of Section 4.1, we obtain

$$k \cdot [J - k \cdot (1 - W)] \cdot P_{k\_RO} = k \cdot [J - (k-1)] \cdot (1 - W) \cdot P_{(k-1)\_RO}$$

or

$$\begin{aligned} k \cdot [J - k \cdot (1 - W)] \cdot P_{k\_RO} = \quad & \{(k-1) \cdot [J - (k-1)] \cdot (1 - W) \\ + \quad & [J - (k-1)] \cdot (1 - W)\} \cdot P_{(k-1)\_RO} \end{aligned}$$

or

$$\begin{aligned} & [k \cdot J - k^2 \cdot (1 - W)] \cdot P_{k\_RO} \\ = \quad & (1 - W) \cdot [J + (J-1) \cdot (k-1) - (k-1)^2] \cdot P_{(k-1)\_RO} \\ & for \quad k = 2, ..., J. \end{aligned}$$

Let's sum all the left hand sides of these equations and let's equate the result to the sum of the right hand side; we have

$$\begin{aligned} & J \cdot X_1 - J \cdot P_{1\_RW} - (1 - W) \cdot X_2 + (1 - W) \cdot P_{1\_RW} \\ = \quad & J \cdot (1 - W) \cdot (1 - P_{J\_RO} + (J-1) \cdot (1 - W) \cdot (X_1 - J \cdot P_{J\_RO}) \\ & -(1 - W) \cdot (X_2 - J^2 \cdot P_{J\_RO}) \end{aligned}$$

After some simplification, we have

$$\begin{aligned} [J - (J-1) \cdot (1 - W)] \cdot X_1 \quad &= \quad J \cdot (1 - W) + (J - 1 + W) \cdot P_{1\_RW} \\ &= \quad J \end{aligned}$$

or

$$X_1 = \frac{J}{1 + (J-1) \cdot W}$$

But

$$1 - \frac{X_1}{J} = \frac{J-1}{J} \cdot P_{1\_RW} + \sum_{k=2}^{J} \frac{J-k}{J} \cdot P_{k\_RO} = ls \cdot P(M)$$

Therefore,

$$P(M) = \frac{1}{ls} \cdot \frac{(J-1) \cdot W}{1 + (J-1) \cdot W}$$