Models for Performance Prediction of Cache Coherence Protocols

Sinisa Srbljic, Zvonko G. Vranesic, Michael Stumm and Leo Budin*
{sinisa, zvonko, stumm}@eecg.toronto.edu
leo@zemris.fer.hr

Technical Report CSRI - 332
July 1995

Computer Systems Research Institute
University of Toronto
Toronto, Ontario
M5S 1A1

*Faculty of Electrical Engineering and Computing
University of Zagreb, Zagreb, Croatia

Abstract

In a modern shared memory multiprocessor, it is possible to support more than one protocol for maintaining cache coherence. Possible candidates might be based on the Write-Back/Invalidate, Write-Through/Invalidate, and Write-Update protocols. *Hybrid* protocols allow the use of different protocols for different data blocks, and *dynamic hybrid* protocols additionally allow for changes in the choice of protocol during the execution of an application.

In this paper, we introduce a set of analytical models for predicting the performance of parallel applications under various cache coherence protocol assumptions. These models can be used at compile time to determine which protocols are to be used for which data blocks, and also to determine when to change protocols in the case of dynamic protocols. Although we focus on tightly-coupled multiprocessor systems, the models apply equally well to loosely-coupled distributed systems such as networks of workstations.

Our models are unique in that they lie between a large body of theoretical models that assume independence and a uniform distribution of memory accesses across processors, and a large body of address-trace oriented models that assume the availability of a precise characterization of interleaving behavior of memory accesses. The former are not very realistic, and the latter are not suitable for compile-time and run-time usage. In contrast, our models use a set of parameters that characterize the access behavior of applications well, and can be obtained with advanced compiler technology.

We present the models and show how the required parameters can be obtained. We assess the accuracy of our models on 15 parallel applications and show that we can predict performance within a 10% margin. As part of this study, we also show the potential advantage of using dynamic hybrid protocols.

Key words: Cache coherence, distributed shared memory, memory access behavior, analytical performance prediction, performance evaluation, dynamic hybrid protocols.

1: Introduction

In a modern shared memory multiprocessors, it is possible to support more than one protocol for maintaining cache coherence. Possible candidates might be based on the Write-Back/Invalidate, Write-Through/Invalidate, and Write-Update protocols. *Hybrid* protocols allow the use of different protocols for different data blocks. For hybrid protocols, it might be possible to specify which protocol to use on a per program, per segment, per page, or on a per cache line basis. *Dynamic hybrid* protocols additionally allow for changes in the choice of protocol during the run time of an application [1].

In this paper, we introduce a set of analytical models for predicting the performance of parallel applications under various cache coherence protocol assumptions. These models can be used at compile time to determine which protocols are to be used for which data blocks, and also, in the case of dynamic protocols, to determine when to change protocols. Each model in the set of models differs in the number and types of parameters it requires. The simplest one, which we refer to as the *core model*, requires a characterization of the type of accesses to each data block, and the probabilities of various accesses. This can easily be obtained using modern compiler technology [2].

The core model is later extended to include interleaving characterization parameters that describe the ordering of accesses performed by different processors on each data block. These parameters can be estimated using compiler technology, but can probably be obtained more accurately by analyzing address traces generated by simulations, or by monitoring previous runs. The latter method assumes the availability of monitoring hardware that allows non-intrusive run-time profiling which is generally non-trivial [3, 4].

The accuracy and usefulness of our models is assessed by comparing the performance predicted by our models with the results of simulated execution for 15 parallel applications, mostly from the SPLASH and SPLASH-2 benchmark suites [5, 6], and also with simulation results reported by others [1, 7]. These comparisons show that even our simplest model is capable of choosing the right cache coherence protocol for all the applications we considered, while the results of our more sophisticated models lie within 10% of simulation results. As a side effect of our studies, we are also able to show the benefits of supporting hybrid and dynamic hybrid protocols, but that the benefits of dynamic protocols are limited to some applications. Although we focus on tightly-coupled multiprocessor systems in this paper, our models can also be easily applied to loosely-coupled distributed memory systems [7-11].

In recent years, many models for predicting the performance of parallel systems have been proposed. They can roughly be classified into two groups based on the information they use to characterize the data access behavior of applications. One group of models assumes that shared data accesses are independent and uniformly distributed across processors and are thus *theoretical* in nature [9, 12-14]. These models, while simple, are inaccurate in their predictions to be useful for our purposes. The second group of models [3, 10, 15-20] is mostly *experimental* in nature in that the models use information from complete memory traces of real applications, or fully simulate the applications [1, 8, 21-24]. They define data access parameters based on the precise interleaving characterization of the memory accesses performed by different processors. For example, they predict performance based on how many different processors perform reads between two writes, or on the number of reads and/or writes performed by one processor. These experimental models are unsuitable for compile-time or run-time usage due to the type and amount of information they require.

The set of models that we propose in this paper is unique in that it lies between the two groups described above, and has, in fact, been developed from experience using models from both groups. The core of our model is theoretical in that it makes no assumptions on the interleaving of memory accesses from different processors. Instead of assuming that accesses to shared data are uniformly distributed across all processors, we consider multiple access patterns to blocks of data which are defined according to the number of processes performing accesses and the type of access they perform. For each of these patterns we define additional parameters such as the probabilities of various accesses. Based on these parameters, we derive one analytical formula for predicting the performance for each pattern and for each basic cache coherence protocol.

The more sophisticated models are refinements of the core model which include parameters that characterize interleaving. These models are then almost experimental in that the parameters are most accurately obtained through address traces, but as we will show, they can easily be averaged and are thus suitable for inclusion in analytical expressions.

Our core model is described in Section 2 and assessed in Section 3. Section 4 extends the core model by introducing interleaving parameters. Dynamic hybrid protocols are considered in Section 5. We close with concluding remarks.

2: The core model

The memory space is partitioned into data blocks of equal size, such as pages or cache lines. The accesses to each data block are then classified into a few predetermined data access patterns according to the number of processors that perform the accesses and according to the type (read, write) of accesses. For example, a data block might have multiple readers and no writers, or it might have multiple readers and a single writer, etc. For each access pattern we introduce parameters, such as the number of processors that access the data block, and the probabilities of the type of accesses. Based on these parameters, we derive analytical formulas for predicting the steady-state costs incurred for each data block, given a particular type of cache coherence protocol.

The core model is described in five steps. Section 2.1 introduces the multiprocessor system we are considering, together with the cache coherence protocols it supports and the system events that can occur. Section 2.2 describes the data access patterns we consider and their parameters. In Section 2.3, we introduce analytical formulas for calculating average costs for each of the access patterns, given the values of their parameters. Section 2.4 describes how the performance of an application can be predicted, given the costs incurred for each data block. Finally, Section 2.5 refines the model by partitioning time into smaller intervals and then calculating the cost incurred at each data block within each time interval. In the discussion it is assumed that all the parameters needed are known. In Section 2.6 we describe how they can be obtained.

To simplify the derivation and to focus on the cost of cache coherence, we assume infinite caches, as was done, for example, in Dubois and Wang's burst model [19, 20]. As done in theoretical models, we also assume that accesses performed by different processors are independent in time, making no assumptions on the interleaving of these accesses. Although accesses in reality do not satisfy this assumption, Section 3 will show that our choice of access patterns and an appropriate choice of time interval for which the data access characterization is performed, results in a sufficiently good prediction of *relative* performance. This means that the core model can be used as a decision making model to chose the best basic protocol.

2.1: The multiprocessor system model

We consider the performance of a multiprocessor that supports three different cache coherence protocols: Write-through and Write-back both of the write-invalidate type, and Update. Table 1 shows all system events that are possible in a multiprocessor system using the three cache coherence protocols. The Write-through protocol can incur three system events: E_2, E_8, and E_9. Each write access updates the memory and the copy in the local cache; all other copies are invalidated. The Write-back protocol can incur six possible events: E_2, E_3, E_5, E_6, E_7, and E_{12}. The first write to a cache line invalidates the copies in all other caches and the copy in the memory. From then on, only the local copy is updated. When a data block is read from a remote cache, then the contents of the data block are also written back to the memory. The Update protocol has three events: E_2, E_{10}, and E_{11}. It keeps the copies in the memory and in all caches coherent after each write. It is also useful to consider uncached operations for which each read and each write proceeds directly to the memory (system events E_1 and E_4).

2.2: Data access patterns and their parameters

Table 2 shows six data access patterns defined by the number of processors that perform reads (loads) and writes (stores)[1]. The meaning of most of these access patterns is straightforward; only the MRSW and SRMW patterns need explanation. In the *Multiple Reader Single Writer* (MRSW) pattern, one processor performs reads and writes, while the other processors perform reads only. The *Single Reader Multiple Writer* (SRMW) pattern involves one processor that performs reads and writes, while other processors perform writes only. To enable a quantitative analysis, we introduce parameters for each access pattern such as the number of processors that perform accesses and the probabilities of these accesses. The required parameters are given in Table 2.

The access patterns and their parameters were chosen to keep the expressions of our models simple, yet provide sufficient accuracy for predicting performance. To avoid excessive complexity, the MRMW pattern assumes that all β processors perform writes with equal probability ρ/β and reads with equal probability $(1-\rho)/\beta$. Similarly, the MRSW and SRMW patterns assume that all β readers read with probability σ, and all β writers write with probability ξ,

1. The concept of classifying data by how it is accessed (i.e. degree of sharing and access mode) has also been used by other researches [8, 25-27], although their classification differs slightly from ours.

Table 1: System events

System event	System Event Description
	Load Instruction
E_1	Read one word from memory
E_2	Read a data block from memory
E_3	Read a data block from remote cache (Dirty copy) and write back to memory
	Store Instruction
E_4	Write one word to memory
E_5	Invalidate[1]
E_6	Read a data block from memory and invalidate other copies
E_7	Read a data block (Dirty copy) from remote cache
E_8	Update the memory and invalidate all other copies
E_9	Update the memory, invalidate all other copies, and read a data block from memory[2]
E_{10}	Update both the memory and all caches
E_{11}	Update the memory and all caches and read a data block from memory
	Ejection of Dirty[3] copy
E_{12}	Write back a data block to memory

1. This event occurs when a write is issued to a cache line in Valid state and ownership must be obtained.
2. This event occurs when a write is issued to a cache in Invalid state; both ownership and data block must be obtained.
3. Invalid, Valid and Dirty states are defined in the standard way for the Write-back/invalidate protocol.

respectively. We assume that the probabilities of accesses performed by different processors are averaged among these processors. Of course, this does not correspond to reality; an analysis of data accesses of real applications shows that the probability ρ/β for MRMW and probabilities σ and ξ for MRSW and SRMW are typically not equal for all β processors, and that the set of processors which perform reads is not equal to the set of processors which perform writes in the case of MRMW. A comparison with simulation results will show (in Section 3) that averaging of the parameters results in sufficiently good predictions while keeping the analytical expressions simple.

2.3: Cost calculation for different data access patterns

The core of the model consists of a set of expressions for calculating the overhead for each access pattern and for each cache coherence protocol. The expressions are based on the steady-state analysis of the multiprocessor system, where the probability of each system event is multiplied by the cost of the event. These products are summed to obtain the steady-state average cost per access for a particular *pattern* (MR, MW, SRSW, MRSW, SRMW, or MRMW) and cache coherence *protocol* (UP - Update, WT - Write-Through, WB - Write-Back, or UC - Uncached):

$$C_{pattern,\ protocol} = \sum_{\text{system event } E_k} c_k \pi_k \qquad (1)$$

where c_k is the cost for the system event E_k, and π_k is the probability of event E_k. In our studies, the cost c_k corresponds to average processor stall time expressed in the number of clock cycles needed to perform event E_k. We note that other metrics may be used, such as the number of packets required in a distributed system.

The probabilities π_k can be derived as follows. We define a sample space consisting of read and write accesses which are treated as random events. It is assumed that they are mutually exclusive and independent in time. Therefore, a specific sequence of accesses can be treated as a sequence of repeated independent trials. The probability of a

Table 2: Data access patterns and parameters

Data Access Patterns		Pattern Parameters	
MR	Multiple Reader	β	number of processors that perform reads
MW	Multiple Writer	β	number of processors that perform writes
SRSW	Single Reader Single Writer[1]	ρ	probability that the access is a write
		$1 - \rho$	probability that the access is a read
MRSW	Multiple Reader Single Writer	β	number of multiple readers
		σ	probability that the access is a read from one of the β multiple readers[2]
		ρ	probability that the access is a write from a single writer
		$1 - \rho - \beta\sigma$	probability that the access is a read from a single writer
SRMW	Single Reader Multiple Writer	β	number of multiple writers
		ξ	probability that the access is a write from one of the β multiple writers[2]
		ρ	probability that the access is a write from a single reader
		$1 - \rho - \beta\xi$	probability that the access is a read from a single reader
MRMW	Multiple Reader Multiple Writer	β	number of processors that perform reads and writes
		ρ / β	probability that the access is a write from one of the β processors[2]
		$(1 - \rho) / \beta$	probability that the access is a read from one of the β processors[2]

1. Single Reader (SR) and Single Writer (SW) patterns are defined as the SRSW pattern with $\rho = 0$ and $\rho = 1$, respectively.
2. To simplify the derivation of expressions for performance prediction, it is assumed that all β processors perform these accesses with equal probability.

specific sequence of accesses is equal to the product of the probabilities of the individual accesses. To obtain the probability of a specific system event, the probabilities of all sequences which result in this event have to be summed.

As an example, we will derive the probability of event E_7 for the Write-back protocol and MRMW pattern. This event is the exclusive read from a remote cache. It occurs if a write from processor *j* follows a write from a different processor *i*. There may be any number of reads from processor *i* between these two writes. Let W_i and W_j denote the writes from processors *i* and *j* and R_i the read from processor *i*. Then, the probability of sequences $W_iR_iR_i,...,R_iW_j$ is

$$p\left(\sum_{z=0}^{\infty} W_i\,\underbrace{(R_i,\ldots,R_i)}_{z\ \text{times}}\,W_j\right) = \sum_{z=0}^{\infty} p(W_i)\,\left(p(R_i)\right)^z p(W_j) = \sum_{z=0}^{\infty}\left(\frac{\rho}{\beta}\right)^2\left(\frac{1-\rho}{\beta}\right)^z \qquad (2)$$

where ρ/β denotes the probability of processor *i* performing a write and the probability of processor *j* performing a write, and $(1-\rho)/\beta$ is the probability of processor *i* performing a read. The sum goes from zero to infinity because any number of reads from processor *i* can be performed between the two writes. Expression 2 is given for only one pair of processors. After summing the probabilities for all possible pairs of processors, which is equivalent to multiplying Expression 2 by $\beta(\beta-1)$ and calculating the sum of the given series, we obtain an expression for π_7 given in Table 3.

Similarly, probability π_3, which is the probability of reading the modified copy from a remote cache, can be derived as

$$\pi_3 = \beta(\beta-1)\left(p\left(\sum_{z=0}^{\infty} W_i\,\underbrace{(R_i,\ldots,R_i)}_{z\ \text{times}}\,R_j\right)\right) = \beta(\beta-1)\sum_{z=0}^{\infty}\frac{\rho}{\beta}\left(\frac{1-\rho}{\beta}\right)^{z+1} \qquad (3)$$

The expressions for all MRMW system events are given in Table 3. The corresponding expressions for the MR, MW and SRSW events can be easily obtained from the MRMW expressions by setting ρ=0, ρ=1, and β=1, respectively. Note that the probabilities of all MR system events which incur costs are equal to zero for all cache coherence protocols, because all accessing caches will have valid copies in the steady state. Similarly, the probabilities of SRSW events which incur overheads using the Write-back protocol are equal to zero, because the accessing cache will own the data block in the steady state and have a local copy in the Dirty state, which enables local execution of reads and writes. Therefore, the average steady-state costs $C_{MR,\,*}$ and $C_{SRSW,\,WB}$ are equal to zero. The expressions for the

Table 3: Probabilities for system events for the MRMW pattern
(π_k is the probability of system event E_k defined in Table 1)

Write-through	Write-back[1]
$\pi_2 = \rho (\beta - 1)(1 - \rho) / (1 + (\beta - 1)\rho)$	$\pi_2 = \rho (\beta - 1)(1 - \rho) / (1 + (\beta - 1)\rho) - \rho(\beta - 1)(1 - \rho) / (\rho + \beta - 1)$
$\pi_8 = \rho - (\beta - 1)\rho^2 / (1 + (\beta - 1)\rho)$	$\pi_3 = \rho (\beta - 1)(1 - \rho) / (\rho + \beta - 1)$
$\pi_9 = (\beta - 1)\rho^2 / (1 + (\beta - 1)\rho)$	$\pi_5 = \rho - (\beta - 1)\rho^2 / (1 + (\beta - 1)\rho) - \rho^2 / (\rho + \beta - 1)$
	$\pi_6 = (\beta - 1)\rho^2 / (1 + (\beta - 1)\rho) - (\beta - 1)\rho^2 / (\rho + \beta - 1)$
	$\pi_7 = (\beta - 1)\rho^2 / (\rho + \beta - 1)$
Update[1]	**Uncached**
$\pi_{10} = \rho$	$\pi_1 = 1 - \rho$ $\pi_4 = \rho$

1. The formulas are derived from the steady-state analysis of the system with infinite cache; therefore, the probabilities of events π_2 and π_{11} for the Update protocol and π_{12} for the Write-back protocol are equal to zero.

MRSW pattern are given in Table 9 and for the SRMW pattern in Table 10 in Appendix A. The expressions are also given in [28-30].

2.4: Predicting the performance of applications

Predicting the performance of an application can be done by first partitioning the data space into data blocks and then evaluating the cost of each data block separately. For each data block it is necessary to determine the access pattern and parameters and then evaluate the prediction $C_{pattern,\ protocol}$ of the average steady-state cost per access to that data block. $C_{pattern,\ protocol}$ is then multiplied by the percentage of accesses to this block, and the products for all blocks are summed to obtain the average cost per shared access for a given application and cache coherence protocol:

$$C_{protocol}^{application} = \sum_{\text{block } i} \frac{\#\text{Block}_i}{\#\text{Appl}} \; C_{pattern\,(i),\ protocol} \tag{4}$$

$C_{pattern(i),\ protocol}$ is the cost for a block i, whose data accesses are classified as *pattern*(i), for the given coherence protocol. The terms $\#Block_i$ and *#Appl* denote the number of accesses to the block i and the total number of accesses performed by the application, respectively.

While we have shown how to predict the performance of applications, it should be noted that this is not necessary for choosing the most appropriate protocol for each block. It is sufficient to evaluate $C_{pattern(i),\ protocol}$ for each data block. We compute $C_{protocol}^{application}$ primarily to be able to assess the quality of our model by comparing the predicted performance of applications with the results of simulations.

2.5: Access pattern characterization using smaller time intervals

The analysis of data access patterns of applications shows that the access pattern characterization of data blocks typically changes in time [27]. For example, a block characterized as SRSW may become MR at a later time. As a result, predicting performance using an access characterization based on complete application runs will lead to inaccurate results, because the parameters do not reflect temporal changes. For the example above, a block that is first SRSW and later MR would simply be characterized as MRSW. Therefore, access pattern characterization and performance prediction should be done for smaller time intervals. If the time needed to run an application is partitioned into smaller intervals, then the performance of the application can be predicted as:

$$C_{protocol}^{application} = \sum_{\substack{\text{block } i \\ \text{time interval } j}} \frac{\#\text{Block}_{ij}}{\#\text{Appl}} \; C_{pattern\,(i,\,j),\ protocol} \tag{5}$$

where the sum includes the predictions for each block and for each time interval. The term $\#\text{Block}_{ij}$ denotes the number of accesses to the block *i* during the characterization interval *j*.

A correct choice of size for the characterization interval is important in order to obtain accurate results with our model and will be discussed further in Section 3.4.

2.6: Determining values of parameters

Our model requires estimates for the parameters β, ρ, σ, and ξ on a per-block and per-interval basis. There are three ways these parameters can be obtained. First, they can be obtained from address traces generated by a simulator. This is how we obtained the parameters that were input into our models (see Section 4). Second, the parameters can be estimated from data obtained from previous runs of the application with the help of monitoring hardware. This implies, however, that hardware exists which can monitor accesses on a per-block basis in a non-intrusive way.

Third, compiler technology can be used to estimate the required parameters. The compiler might split the application into a group of statements or regions [2]. Data dependence analysis can then be performed for each region in order to estimate the access pattern and the associated parameters. We believe that the required parameters can be estimated relatively easily, because it is not necessary to predict the ordering among accesses, but rather just the number of processors that perform the accesses and the probabilities for the types of accesses.

3: Assessing the core model

In this section, we assess the quality of our core model by comparing the performance predicted by the model to the performance determined by simulating a real system. Overall, we analyzed the performance of 15 applications, mostly from the SPLASH [5] and SPLASH-2 [6] benchmark suites. Here we present the results obtained from three representative applications. The first two are BARNES (512 particles) and MP3D (25000 molecules, 5 steps, test.geom), both from SPLASH, and the third is LU decomposition (100×100 matrix).

We describe the simulated system in Section 3.1, and then the data access pattern characterization of each application in Section 3.2. In Section 3.3, we discuss the performance of the applications obtained through simulation and compare them to the predictions generated by our model. This is followed by a discussion on the best choice of characterization interval size in Section 3.4.

We show that with an appropriate size of characterization interval we are able to accurately predict the relative performance, making the core model useful in choosing the best basic protocol. However, the comparison with simulation results shows inaccuracy in the prediction of *absolute* values. Section 4 describes extensions to the core model that improve the accuracy of the performance values predicted.

3.1: Simulation details

In order to obtain results for different architectural configurations, we simulated two bus-based multiprocessor systems: one having 8 processors, and another having 16. To obtain realistic cost estimates, we simulated the systems with MIPS R4400 or R10000 processors [31, 32]. One 64-bit word can be transferred in one clock cycle in the 8-processor system, and two such words can be transferred in one clock cycle in the 16-processor system. The costs for the system events are given in Table 4. These costs are given in clock cycles which correspond to processor stall time. Some of the parameters have two terms. The first is a constant that accounts for the average number of clock cycles spent on bus arbitration, memory and cache latency, and processor response time[1]. The second term represents the number of clock cycles needed to transfer a data block. In this section and the next, we present results for the 8-processor system. Section 5 will present results for the 16-processor system.

1. We do not take bus and memory contention into account. We use the same constant that does not depend on the amount of traffic. For write-invalidate protocols, the comparison with simulation results shows that our performance prediction for large blocks (larger than 1K bytes) is slightly optimistic, because the data accesses spend more time waiting for common resources such as the bus and the memory due to the higher traffic caused by the larger blocks. This could be corrected if appropriate constants are chosen for specific block sizes. For uncached operations and the Update protocol the amount of traffic is the same regardless of block size. This enables accurate prediction for all block sizes by using the same constants.

Table 4: System events costs
(c_k is the cost of system event E_k
denoting processor stall time in clock cycles;
h - Block size in bytes)

8-Processor Multiprocessor[1] 64-Bit Data Bus	16-Processor Multiprocessor[2] 128-Bit Data Bus
Load Instruction	
$c_1 = 12$	$c_1 = 27$
$c_2 = 10+h/8$	$c_2 = 26+h/16$
$c_3 = 15+h/8$	$c_3 = 29+h/16$
Store Instruction	
$c_4 = 5$	$c_4 = 10$
$c_5 = 20$	$c_5 = 30$
$c_6 = 22+h/8$	$c_6 = 32+h/16$
$c_7 = 15+h/8$	$c_7 = 29+h/16$
$c_8 = 20$	$c_8 = 30$
$c_9 = 22+h/8$	$c_9 = 32+h/16$
$c_{10} = 20$	$c_{10} = 30$
$c_{11} = 22+h/8$	$c_{11} = 32+h/16$
Ejection of Dirty copy	
$c_{12} = 4+h/8$	$c_{12} = 10+h/16$

1. The constants are determined based on the technical specification of the MIPS R4400 processor [32].
2. The constants are determined based on simulation results of the University of Toronto NUMAchine multiprocessor [34].

3.2: Data access pattern characterization of applications[1]

Figure 1 shows the data access pattern characterizations of three applications: BARNES, MP3D and LU as obtained by processing address traces generated by the MINT program [33]. We give the characterization of shared data accesses for various data block and time interval sizes[2]. Our graphs involve seven different block sizes. The largest size is 1M bytes and the next three sizes correspond to pages of 8K, 4K, and 1K bytes. The remaining three sizes correspond to typical cache lines of 256, 128, and 64 bytes. We use four different characterization interval sizes 10^6, 10^5, 10^4 and 10^3 processor cycles.

In the rest of the paper we will say that data accesses are of a particular type according to the type of data access pattern in which they occur. For example, data accesses occurring in the MR pattern are said to be of MR type. Each access type has its own graph, and the results are presented in terms of percentages of total accesses. For a given block and characterization interval size, the percentages of all access types add up to 100. For example, the graph MR shows the percentage of accesses to all data blocks for which the data access patterns for a given characterization interval and block size are classified as MR, i.e. the percentage of the accesses which are of the MR type. We only show those access types that are significant for a particular application. For the largest block size (1M bytes) and characterization interval (10^6 processor cycles), corresponding to the lower left hand corner of the graphs, one intuitively expects that shared data accesses will be mostly of MRMW type. Other points correspond to smaller blocks and characterization intervals in which other shared data access types become more probable. For extremely small intervals (not considered in our graphs) there will typically be at most one access per interval, in which case all shared data accesses will be of SRSW type.

Figure 1*a* depicts the access characterization for BARNES. We can see that even with a large block and charac-

1. In this paper we consider sharing of data blocks. Since we are assessing different cache coherence protocols, we are not interested in the kind of sharing, whether it is true or false. Both kinds of sharing have the same affect on performance of cache coherence protocols.
2. Brorsson and Stenstrom use similar graphs to visualize the data access pattern characterization [25].

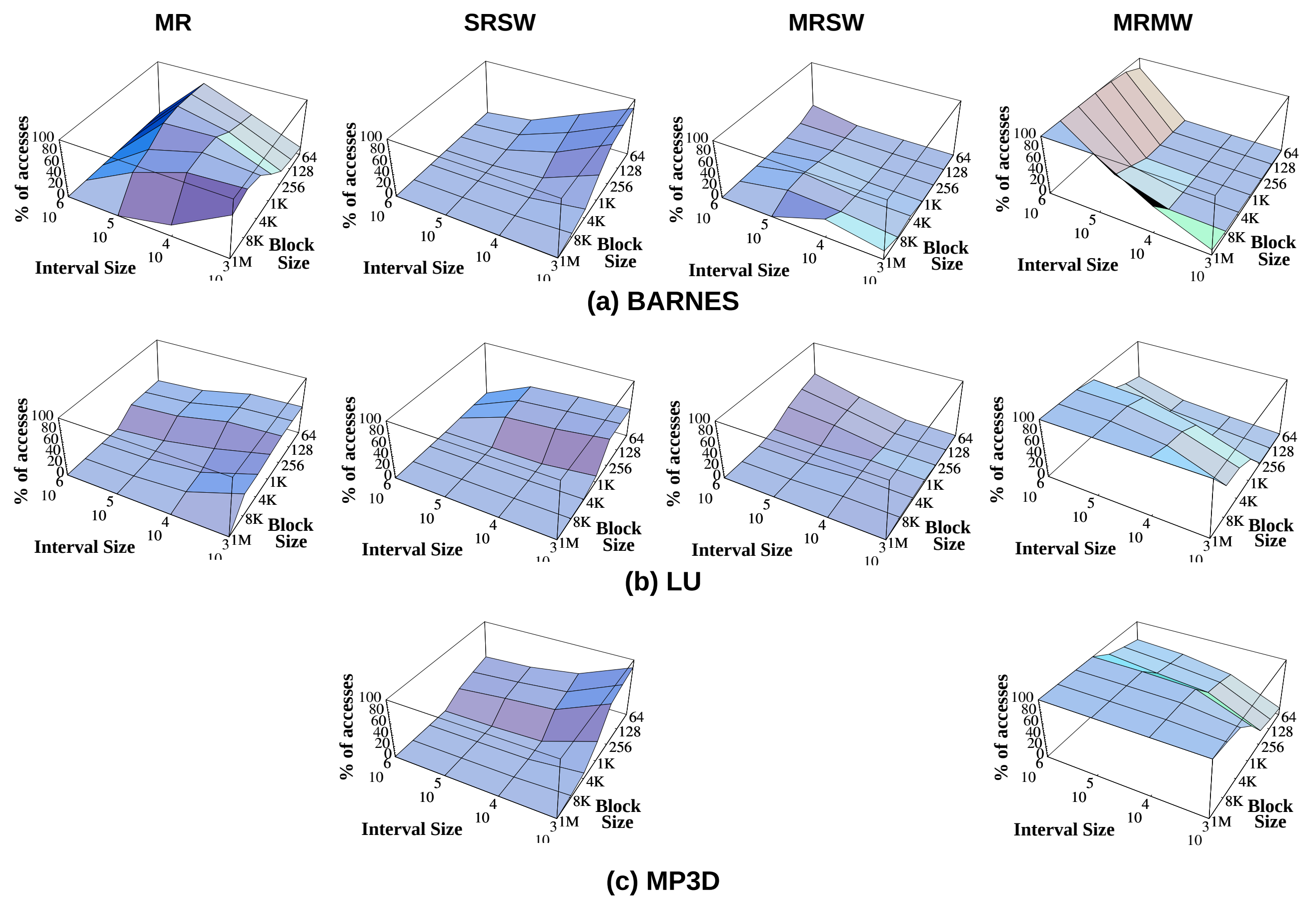


Figure 1: Data access pattern characterization
(Interval size in processor cycles; Block size in bytes)

terization interval sizes, almost all of the shared data accesses are of MR type. The percentage of MR accesses decreases as the characterization interval size decreases, while the percentage of SRSW accesses increases correspondingly. In this case, the MR type mostly becomes SR type, which is included in the SRSW type.

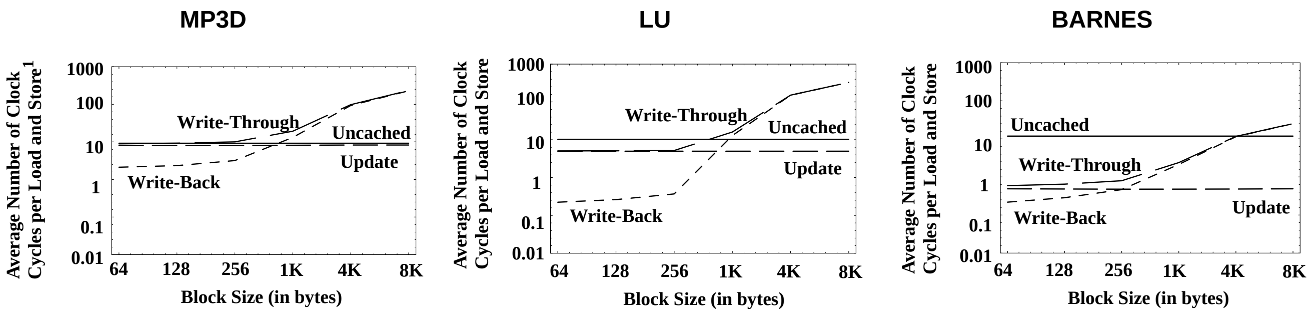
Figure 1*b* shows the access characterization for the LU application. As the block size decreases, the percentage of shared accesses of the SRSW and MR types increases.

The access characterization for MP3D is presented in Figure 1*c*. It shows that there are only two significant types of shared data accesses for this application: MRMW and SRSW. If we decrease the characterization interval size, then the percentage of MRMW type accesses decreases, while the percentage of SRSW accesses increases.

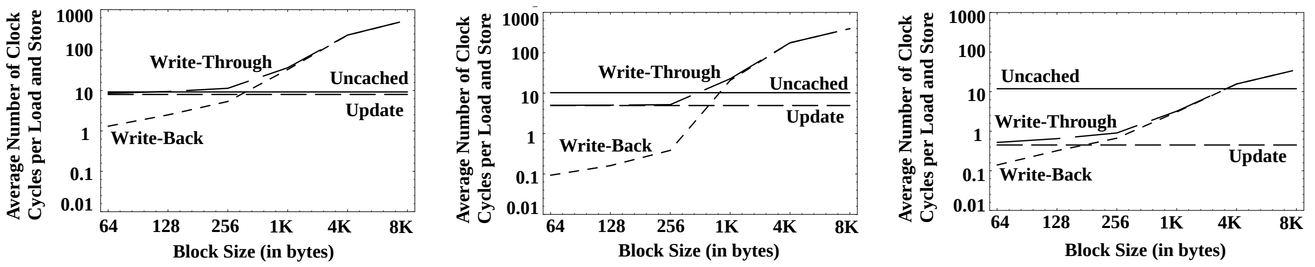
3.3: Simulation results and the quality of our core model

Figure 2 shows the cost per access as predicted by our model and as determined by simulation runs for several cache coherence protocols. The first row shows the results of simulation, while the other two rows show the results of analytical prediction using two different sizes of characterization intervals (10^3 and 10^5 processor cycles). The costs are given as averaged values per shared access, which enables both a comparison of simulation and analytical results for a given application (by comparing graphs in the same column), as well as a comparison of results for different applications (by comparing graphs in the same row).

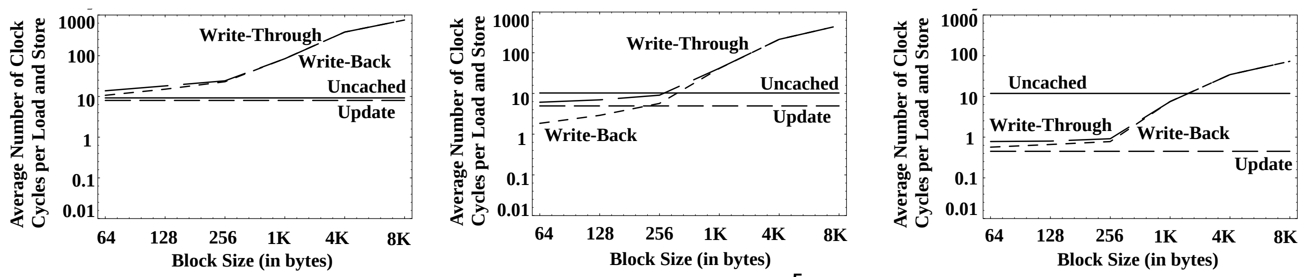
We begin with the leftmost graph in Figure 2*a*, which shows the performance for MP3D obtained by simulation. For small blocks, the Write-back protocol outperforms other protocols because most of the accesses are of the SRSW type (see Figure 1*c*). Only the first write incurs a cost for the Write-back protocol, while each write incurs a cost for the other two cache coherence protocols. For uncached accesses, each read and write incurs a cost, so the performance for these accesses is the same for all block sizes. Since the percentage of writes is very high in MP3D



(a) Performance determined by simulation (Cache size = infinite)



(b) Analytical prediction: Interval size = 10^3 processor cycles



(c) Analytical prediction: Interval size = 10^5 processor cycles

Figure 2: Comparison of analytical and simulation results (8-processor system)

1. The number of load and store instructions is the same for all protocols for a given application.

(about 40%), and all other accesses that are not of SRSW type are of MRMW type, the difference between Update, Write-through, and uncached operations is small. The Update protocol outperforms the other protocols for large blocks, because the read/write ratio and the cost for updating do not depend on the block size[1]. In contrast, the invalidation based protocols become more expensive with larger blocks, because of the cost of block transfer and also because the probability of a block being shared increases. Also, the probability of reading or writing invalidated data becomes higher because most of the accesses are of MRMW type (see Figure 1*c*). Therefore, Write-back and Write-through protocols give poor performance for large blocks.

Figure 2*a* also shows the simulated performance for LU decomposition. As in MP3D, the Write-back protocol performs best for small blocks. Since data accesses that are not of the SRSW type are of the MR type, and because of the lower percentage of writes (about 24%), the cache hit rate is higher for LU than for MP3D. The low percentage of writes causes a higher cost for uncached accesses, because the cost of a read is higher than the cost of a write. For large blocks, Update performs better than the other basic protocols.

The rightmost graph in Figure 2*a* shows the simulated performance for the BARNES algorithm. The difference between the performance for cached and uncached accesses is large because of the low percentage of writes (about 2%), and because most accesses are of SRSW type (see Figure 1*a*). The Write-back protocol outperforms the other basic protocols for small blocks, while the Update protocol outperforms the other protocols for large blocks. These three applications indicate that the trend of processor manufactures to primarily support the Write-back protocol is a

1. Performance of the Update protocol depends only on the read/write ratio and the cost of updating.

good one.

As to the quality of our model, we can see from Figure 2 that the results with the interval size set to 10^3 processor cycles (Figure 2*b*) correspond more closely to those produced by the simulator (Figure 2*a*) than the results generated when the interval size is set to 10^5 (Figure 2*c*). Hence, the quality of results using our model depends on the size of the characterization interval if either Write-through or Write-back protocol is used. (The size of the characterization interval does not affect the results if the applications run uncached operations or if the Update protocol is used, because the results depend only on the probability of writes, as can be seen from Table 3).

3.4: Appropriate choice for characterization interval size

Choosing an appropriate interval size is important in order to accurately assess the performance of the Write-through and Write-back protocols. If the interval is too large, then the predicted performance will be pessimistic, and if it is too small, then the predicted performance will be optimistic. The reason for overestimating the cost per access when the characterization interval is too large is that the core model assumes that all reads and writes are independent in time. Hence, the probability of a sequence of two accesses to the same block from the same processor is much lower than the probability of two accesses to the same block from different processors. However, in practice, accesses to blocks tend to come in bursts from a single processor at a time [17-20]. This can also be seen in Figure 1, where the percentage of SRSW accesses increases when the characterization interval size is reduced. The probability of two consecutive accesses being from the same processor is higher than the probability assumed by the core model, and the costs of sequences of accesses from the same processor, which are mostly zero, are lower on average than the costs for sequences from different processors, which are non-zero if at least one of the processors performs writes. Thus, the performance predicted for larger characterization intervals will be pessimistic.

On the other hand, too small an interval size will underestimate the costs, because the number of accesses per interval will be too small for the transient costs to be negligible. For example, in the case of the Write-back protocol, the predicted cost for the SRSW pattern is $C_{SRSW,WB} = 0$, as explained in Section 2.3. This is a good approximation if the interval contains a large number of accesses where the cost of reading a block once and/or obtaining exclusivity can be ignored because it will be amortized over a large number of accesses. The smaller the size of the characterization interval, the more important transient effects become, making the results of our model too optimistic.

The choice of characterization interval size also depends on the data block size. For large blocks, data accesses behave, for the most part, as if they were independent in time since long bursts from a single processor become less likely. Moreover, the effects of transients are lower, because the average number of accesses per block is higher than in the case of smaller blocks. Figure 2 confirms that the predictions for large data blocks are close to simulation results for both characterization intervals and shows that relative performance is correctly predicted.

While the choice of characterization interval is not critical for large data blocks, it is important to choose an appropriate characterization interval in order to minimize the effects of both access bursts and transients for 256-byte and smaller data blocks. A more precise analysis of 256-byte blocks shows that for a characterization interval of 10^3 cycles, the average number of accesses per interval is about 80 for LU and about 30 for MP3D and BARNES. Our analysis of other applications shows that relative performance can be accurately predicted for all characterization intervals which have between 20 and 100 accesses per block. However, there will be inaccuracies in the prediction of absolute values, because of the effects of access bursts and transients.

In summary, it is important to use a characterization interval size that minimizes the effects of both access bursts and transients. Access bursts argue for using smaller characterization intervals so that accesses appear to be independent, while transients argue for using larger characterization intervals.

4: Extending the core model

This section describes the extensions to the core model that are necessary to improve the absolute values predicted. The extensions assume that some information on the ordering of accesses by the different processors is available. For the MW, MRSW, SRMW, and MRMW patterns and the write-invalidate protocols, performance is significantly affected by the order of accesses from different processors. (It does not affect the performance of the Update protocol or uncached operation; nor does it affect the performance of the SRSW and MR patterns when the write-invalidate protocols are being used.)

We separately analyze two cases depending on whether there is more or less interleaving of accesses by differ-

Table 5: Probabilities for system events for the MRMW pattern with the λ parameter
(π_k is the probability of system event E_k defined in Table 1)

Write-through	Write-back
$\pi_2 = \rho(\beta - 1)(1 - \rho) / (\lambda\rho + 1 + (\beta - 1)\rho)$	$\pi_2 = \rho(\beta - 1)(1 - \rho) / (\lambda\rho + 1 + (\beta - 1)\rho) - \rho(\beta - 1)(1 - \rho) / ((\lambda+1)\rho + \beta - 1)$
$\pi_8 = \rho - (\beta - 1)\rho^2 / (\lambda\rho + 1 + (\beta - 1)\rho)$	$\pi_3 = \rho(\beta - 1)(1 - \rho) / ((\lambda+1)\rho + \beta - 1)$
$\pi_9 = (\beta - 1)\rho^2 / (\lambda\rho + 1 + (\beta - 1)\rho)$	$\pi_5 = \rho - (\beta - 1)\rho^2 / (\lambda\rho + 1 + (\beta - 1)\rho) - (\lambda + 1)\rho^2 / ((\lambda + 1)\rho + \beta - 1)$
	$\pi_6 = (\beta - 1)\rho^2 / (\lambda\rho + 1 + (\beta - 1)\rho) - (\beta - 1)\rho^2 / ((\lambda+1)\rho + \beta - 1)$
	$\pi_7 = (\beta - 1)\rho^2 / ((\lambda+1)\rho + \beta - 1)$

ent processors than that assumed by the core model. To improve our prediction for the case where interleaving is less than the amount assumed by the core model, we introduce the parameter λ, where $\lambda \geq 0$. The λ parameter denotes that the probability of the next access being from the same processor is λ+1 times higher than the probability of the next access being from a different processor (the term λ+1 is chosen to simplify the derivation of expressions). To predict the performance for the case where interleaving is greater than the amount assumed by the core model, we introduce the parameter η, where $\eta \geq 0$. The η parameter denotes that after an access from processor *i*, the probability that the next access will be from a different processor is η+1 times greater than the probability of the next access being from the same processor.

At the end of the section, we show that the accuracy of the model is retained when the same set of *averaged* parameters are used for each set of data blocks and characterization intervals (for given access type), instead of using separate parameter values for each block and interval.

4.1: Adding the λ parameter

To include the λ parameter, we simply redefine the probability π_k of event E_k occurring. For example, for the MRMW case, once we know that a write is performed by processor *i*, then the probability of a write by the same processor *i* will be ρ(λ+1)/(λ+β), the probability of a read by the same processor *i* will be (1-ρ)(λ+1)/(λ+β), the probability of a write by another processor will be ρ/(λ+β), and the probability of a read by another processor will be (1-ρ)/(λ+β). This insight allows us to redefine the probabilities of Table 3. For example, the probability of event E_7 for the MRMW pattern using the Write-back protocol can be calculated as:

$$\pi_7 = \beta(\beta - 1) \sum_{z=0}^{\infty} \frac{\rho}{\beta} \left(\frac{(1-\rho)(\lambda+1)}{\lambda+\beta} \right)^z \frac{\rho}{\lambda+\beta} \tag{9}$$

Note that the probability of the write W_i is ρ/β because all β processors have an equal probability of performing the first access in the sequence $W_iR_iR_i,...,R_iW_j$ in our assumptions. Once the access is performed by processor *i*, then the probability of this processor subsequently accessing the same block before another processor does so will be λ+1 times higher then the probability of subsequent accesses from other processors.

The expressions for the other probabilities can be derived in a similar fashion. Table 5 gives the expressions for all probabilities that occur in the MRMW pattern for the Write-through and Write-back protocols. The expressions for MR, MW and SRSW patterns can again be derived from the expressions for the MRMW pattern by setting ρ=0, ρ=1, or β=1. The expressions for the MRSW and SRMW patterns are given in Tables 11 and 12 in Appendix A.

4.2: Estimating the λ parameter from a memory trace

In order to estimate λ, we need to measure or estimate the average number of successive local accesses in the characterization intervals. Accesses are considered local if they do not result in overhead. For example, a read from a Shared or Dirty state is considered to be a local access, while a read from an Invalid state, in which a valid copy must be obtained from either a memory or a remote cache is an example of a remote access.

The average numbers of successive local accesses to a block can be measured from a memory trace by counting the number of local and remote accesses. This can be done by introducing an *n*-bit state variable in which each bit

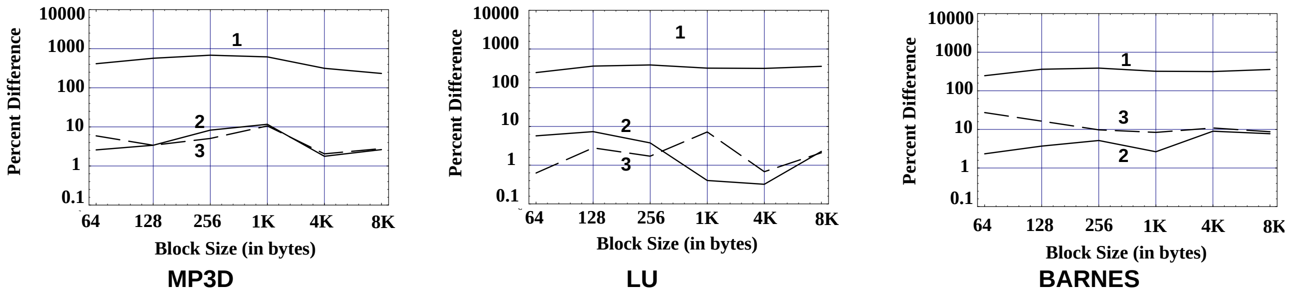


Figure 3: Write-back protocol: Percent difference between simulation results and the analytical prediction with and without included λ parameter (8-processor system, Interval size = 10^5 processor cycles)
1 - Parameter $\lambda = 0$
2 - Performance prediction based on prediction for each characterization interval and for each block separately - λ parameter included
3 - Performance prediction based on averaged parameters - λ parameter included (see Table 6)
(Write to Shared state is considered as local access for 8K and 4K byte block size, otherwise is remote)

corresponds to one of the *n* processors. On each read, the bit corresponding to the accessing processor is ORed with the old value of the state, while each write resets all bits except the one corresponding to the writing processor. A read is considered local if the bit in the *n*-bit state variable corresponding to the reading processor is already set, and it is considered remote otherwise. To determine whether a write is considered local, we consider two variants for the following reasons. For large block sizes the cost of invalidation is small relative to the cost of transferring a data block, and almost insignificant, while for small block sizes the cost of invalidation and the cost of transfer are almost the same. To account for this difference we treat writes to the data blocks in Shared state either as local accesses or as remote accesses depending on whether the data block is large or small. If the write to Shared state is considered local, then we check whether the appropriate bit is already set, as in the read case. If the write to Shared state is considered remote, then we check whether the appropriate bit is set while all other bits are zero. By dividing the overall number of accesses by the number of remote accesses, we can calculate the average number of successive local accesses to the block.

Given the average number of successive local accesses, we can now derive λ. It can be calculated from the probabilities of system events, by setting the average number of successive local accesses to the reciprocal of the probability that an access will be remote and solving for λ. If the writes to blocks in Shared state are assumed to be remote, then the probability that an access will be remote is equal to the sum of the probabilities π_2, π_3, π_5, π_6 and π_7 for the Write-back protocol given in Table 5. Otherwise, if the writes to the Shared state are assumed to be local, then the probability that an access will be remote can be calculated in a similar way except that probability π_5 is excluded form the given sum. Note that at this point, λ is the only unknown parameter; all other parameters, including the value of the average number of successive local accesses are known. For the SRMW pattern we always assume that writes to data blocks in Shared state are local, because our expressions become too complex otherwise. This should not pose a problem, because the number of SRMW accesses tends to be very low.

We have also tried two more intuitive and simpler methods to estimate λ. One is based on the average number of successive accesses performed by one processor, similar to the average burst size in Dubois and Wang’s burst model [19, 20]. Estimating λ using this method does not improve the predictions significantly, except for applications with a high degree of migratory data access sharing (for example, MP3D). The reason for this is that during the periods when the data block is only read, the processors access their own copies simultaneously producing a large interleaving of reads from different processors. The interleaved reads do not increase the coherence overhead because data is not changed. Since it does not increase the overhead, this increased interleaving should not be taken into account when estimating the λ parameter. To exclude the interleaving imposed by these reads, we tried a the method based on the average number of writes performed by one processor, similar to the length of a write-run in Eggers and Katz’ write-run model [17, 18], except that the write-run is terminated only by writes from other processors, and not reads as in the original model. Estimating λ using this method also does not improve the predictions significantly. Since these methods are not suitable for all applications, we do not consider them further in this paper.

Table 6: Averaged pattern parameters
(8-processor system, Block size = 64 bytes; Interval size = 10^5)
The λ parameter is estimated by the average number of successive local accesses with a write to Shared state being defined as a remote access except for SRMW pattern, in which case is defined as a local access

Application	BARNES		LU		MP3D	
Data Access Pattern	% of accesses	Pattern Average Parameters	% of accesses	Pattern Average Parameters	% of accesses	Pattern Average Parameters
MR	**81.843**		**27.733**		**0.247**	
MW	**0.007**	**β = 2.000, λ = 2.098**	**NA**		**0.002**	**β = 8.000, λ = 0**
SRSW	**11.256**	**ρ = 0.067**	**40.659**	**ρ = 0.499**	**41.640**	**ρ = 0.292**
MRSW	**2.551**	**ρ = 0.191, σ = 0.153, β = 2.678, λ = 7.575**	**20.153**	**ρ = 0.068, σ = 0.108, β = 6.961, λ = 145.524**	**0.030**	**ρ = 0.072, σ = 0.125, β = 5.537, λ = 1.718**
SRMW	**0.100**	**ρ = 0.431, ξ = 0.117, β = 1.000, λ = 0.501**	**0.221**	**ρ = 0.0007, ξ = 0.077, β = 4.522, λ = 0.487**	**NA**	
MRMW	**4.240**	**ρ = 0.219, β = 6.026, λ = 59.398**	**11.232**	**ρ = 0.263, β = 4.780, λ = 84.513**	**58.078**	**ρ = 0.477, β = 6.489, λ = 43.254**

4.3: Accuracy of the extended model

Figure 3 shows the difference between the simulation results and the predictions of our model for the Write-back protocol. The figure shows the absolute difference as a percentage of the simulation results. Curve 1 shows the difference between the prediction of our core model that does not take the λ parameter into account and the simulation results. The difference is quite large, partly because of the relatively large characterization interval of 10^5 processor cycles used. This difference was also visible in Figure 2. A large interval was chosen in order to emphasize the improvement in accuracy by using the λ parameter.

Curve 2 shows the difference between the simulation results and the prediction of our model that takes the λ parameter into account. The difference is quite small and usually less then 10%. In calculating λ, a write to a data block in Shared state was considered local for large blocks (4K - 8K) and it was considered remote for the other block sizes.

4.4: Using averaged parameters

While Curve 2 of Figure 3 was generated with a version of our model that uses a separate set of parameter values for each block and each characterization interval, Curve 3 considers a variation where the same averaged set of parameters is used for each block and interval. That is, the parameters are collected from the memory trace and averaged within each of the six access patterns of Table 2. Then, the averaged parameters are used to predict the performance for each block and for each characterization interval. For example, Table 6 lists the averaged parameters for the case of a 10^5-cycles characterization interval and 64-byte block size. They were calculated as a weighted sum of the parameters of all blocks and characterization intervals whose access patterns are of the same type. The weights were calculated by dividing the number of accesses to a particular block and characterization interval by the number of accesses to all blocks and characterization intervals that had access patterns of the same type.

It is interesting to note that performance prediction using only a small number of averaged parameters can be as precise as the performance prediction driven by an entire memory trace. This characteristic enables profiling and performance prediction for large applications for which it is impractical to store the whole memory trace in order to run a simulation.

Figure 3 shows the results for the Write-back protocol only. A comparison of simulation results with the predictions of our model for other protocols also shows good agreement. It should be noted, however, that the performance prediction for dynamic hybrid protocols (discussed in Section 5) cannot be done based on averaged parameters, but must be done with parameters for each block and characterization interval.

Table 7: Probabilities for system events for the MRMW pattern with the η parameter
(π_k is the probability of system event E_k defined in Table 1)

Write-through
$\pi_2 = (\eta+1)\rho(\beta-1)(1-\rho) / (\rho(\beta-1)\eta+ (1-\rho)\eta+1+(\beta-1)\rho)$
$\pi_8 = \rho - (\eta+1)(\beta-1)\rho^2 / (\rho(\beta-1)\eta+(1-\rho)\eta+1+(\beta-1)\rho)$
$\pi_9 = (\eta+1)(\beta-1)\rho^2 / (\rho(\beta-1)\eta+(1-\rho)\eta+1+(\beta-1)\rho)$
Write-back
$\pi_2 = (\eta+1)\rho(\beta-1)(1-\rho) / (\rho(\beta-1)\eta+(1-\rho)\eta+1+(\beta-1)\rho) - (\eta+1)\rho(\beta-1)(1-\rho) / ((\beta-1)\eta+\beta+\rho-1)$
$\pi_3 = (\eta+1)\rho(\beta-1)(1-\rho) / ((\beta-1)\eta+\beta+\rho-1)$
$\pi_5 = \rho - (\eta+1)(\beta-1)\rho^2 / (\rho(\beta-1)\eta+(1-\rho)\eta+1+(\beta-1)\rho) - \rho^2/((\beta-1)\eta+\beta+\rho-1)$
$\pi_6 = (\eta+1)(\beta-1)\rho^2/ (\rho(\beta-1)\eta+(1-\rho)\eta+1+(\beta-1)\rho) - (\eta+1)(\beta-1)\rho^2 / ((\beta-1)\eta+\beta+\rho-1)$
$\pi_7 = (\eta+1)(\beta-1)\rho^2 / ((\beta-1)\eta+\beta+\rho-1)$

4.5: Taking transients into account

While the λ parameter allows the use of large characterization intervals in our model, it is also necessary to take into account transient effects for small characterization intervals (10^3 processor cycles or less) and for small blocks (256 bytes or less). The transient cost is particularly important for blocks whose data accesses for a given characterization interval are of MR or SRSW type. Transient costs can be determined by multiplying the number of remote accesses by the cost of the remote accesses (see Table 4). Two *n*-bit state variables can be used to determine the number of remote accesses from an address trace: one for the Update protocol and one for the write-invalidate protocols. The procedure for the write-invalidate protocols is the same as that described in Section 4.2 for the case where writes to the Shared state are considered remote. The procedure for the Update protocol is different only in that the appropriate bit is set for both reads and writes; the remaining bits are not cleared for writes.

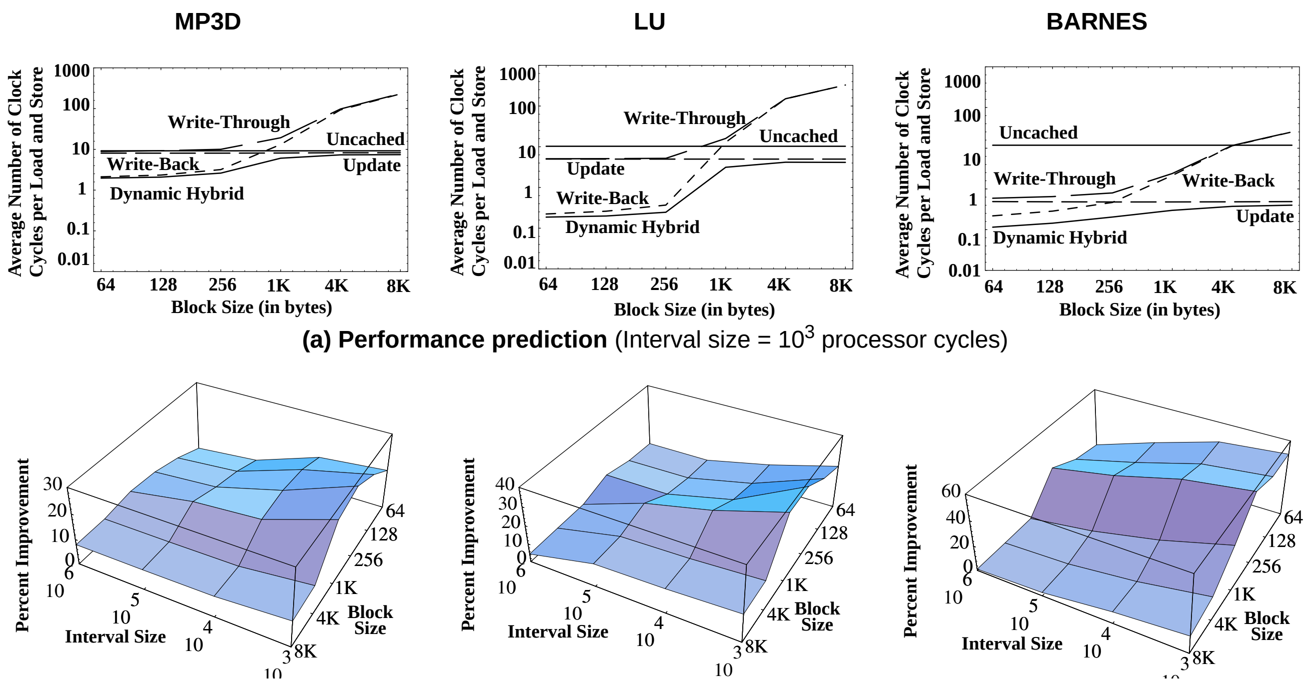
4.6: Adding the η parameter

Even after including transient costs, there still remain some differences between the simulation and the results of our model. The reason is that for small characterization intervals and small blocks interleaving is higher in practice for the MW, MRSW, SRMW, and MRMW accesses than assumed by our model. To better handle the higher amount of interleaving, we introduce the parameter η. This parameter is incorporated into the core model similar to the way the parameter λ was added. The parameter η can be estimated from the memory trace using methods similar to the ones used for estimating λ. In the new sample space, the probability of an access from the same processor is then defined to be $\eta+1$ times lower than the probability of an access from another processor. Table 7 shows the probabilities for the events of Table 2 for the MRMW pattern with the η parameter. As before, the expressions for MR, MW, and SRSW can be derived directly from the expressions for the MRMW pattern, while the expressions for the MRSW and SRMW patterns are given in Tables 13 and 14 in Appendix A.

With the addition of λ and η in our model, the differences between the results generated by simulation and our model are less then 10% for all characterization intervals and block sizes.

5: Dynamic hybrid protocols

Dynamic hybrid protocols can dynamically change the cache coherence protocol during the execution of an application. They can use different basic cache coherence protocols for different blocks and for different characterization intervals. In this section we discuss the predicted performance of dynamic hybrid protocols for a variety of applications from the SPLASH [5] and SPLASH-2 [6] benchmark suites. In particular, we show how two parameters, the block size and the frequency at which the coherence protocol is chosen, affect performance of the dynamic hybrid



(a) Performance prediction (Interval size = 10^3 processor cycles)

(b) The percent improvement for dynamic hybrid protocol

Figure 4: Analytical prediction
(8-processor system)

protocol.

5.1: Performance prediction for dynamic protocols

A simple rearragement of Equation 5 of Section 2.5 enables us to calculate the performance of dynamic hybrid protocols. Instead of using the cost expression for a single basic protocol for all characterization intervals, we can choose the cost expression of the basic protocol that for the characterization interval and block gives the lowest cost. In this manner, we can predict the performance of the dynamic hybrid protocol that chooses the best basic protocol for each characterization interval and block as:

$$C_{hybrid} = \sum_{\substack{\text{block } i \\ \text{time interval } j}} \frac{\#\text{Block}_{ij}}{\#\text{Appl}} \; min_{protocol}\left(C_{pattern\,(i,\,j)\,,\,protocol}\right) \tag{6}$$

5.2: Improvement for hybrid protocols

The improvement attained using dynamic hybrid protocols is determined by comparing it with the basic protocol that gives the best results for a given size of the data block and for a given characterization interval:

$$C_{best} = min_{protocol}\left(C_{protocol}\right) \tag{7}$$

The improvement is then calculated as

$$Improvement = 100 \times \left(C_{best} - C_{hybrid}\right) / \left(C_{best}\right) \tag{8}$$

Table 8: The percent improvement for dynamic hybrid protocols for SPLASH-2 applications
(16-processor system; Interval size in processor cycles)
All applications are run with SPLASH-2 default values
except Barnes (1K particles) and LU Factorization (256×256 matrix).

Application	Interval Size	Block size					
		64 bytes	**128** bytes	**256** bytes	**1K** bytes	**4K** bytes	**8K** bytes
Radix	10^3	53.42	66.79	73.17	85.37	93.24	93.00
	10^6	1.76	18.09	34.85	70.26	89.16	88.86
Barnes	10^3	50.13	53.38	60.54	77.51	89.37	93.84
	10^6	19.24	15.81	17.81	38.01	76.68	86.98
Water-Spatial	10^3	35.18	43.76	49.42	54.11	84.67	92.09
	10^6	27.79	27.03	21.36	10.12	76.72	89.86
FMM	10^3	33.77	36.76	41.35	56.52	85.35	91.64
	10^6	10.67	7.98	6.58	14.93	61.22	76.06
Ocean (contiguous)	10^3	31.96	36.35	40.76	45.57	75.38	85.71
	10^6	3.60	4.03	4.53	7.20	14.21	21.91
Raytrace	10^3	30.13	38.21	47.70	70.92	84.47	86.50
	10^6	27.33	35.73	45.12	66.72	80.03	82.23
Cholesky	10^3	8.18	17.05	29.90	67.74	84.51	88.93
	10^6	4.68	12.00	23.10	62.09	69.31	69.64
Water-Nsquared	10^3	11.73	15.32	24.29	41.80	76.40	86.00
	10^6	3.61	3.02	2.95	4.14	58.02	72.77
Ocean (noncontiguous)	10^3	29.46	29.85	33.49	51.34	43.91	43.09
	10^6	5.42	5.73	9.55	34.17	25.56	25.51
FFT	10^3	15.82	21.56	27.41	33.83	36.41	41.75
	10^6	2.43	3.73	5.31	8.10	12.67	29.72
LU (contiguous)	10^3	5.96	14.65	22.26	27.88	43.17	70.82
	10^6	0.95	1.49	2.63	5.61	25.47	31.13
LU (noncontiguous)	10^3	5.66	6.72	5.30	34.18	65.78	28.04
	10^6	1.49	2.31	0.09	10.63	60.68	13.72

Figure 4*a* shows the performance of MP3D, LU, and Barnes, as predicted by our extended model which includes the λ parameter, the η parameter, and transients. Of the basic protocols, the Write-back protocol performs best for small data blocks, while Update performs best for larger data blocks, as was discussed in Section 3.3. The figure also shows the predicted performance of the dynamic hybrid protocol. Its performance is slightly better than performance of the Write-back protocol for small blocks, and it is slightly better than performance of the Update protocol for large blocks. We assume here that the best protocol is chosen at the beginning of each characterization interval and not changed during the interval. We also assume that the protocol can be changed at no cost; therefore, our results represent an upper bound on the improvement one might expect from a real system.

Figure 4*b* shows the improvement for the dynamic hybrid protocol. The improvement is given for different sizes of characterization intervals and blocks. As one would expect, the improvement for the dynamic hybrid protocol is greater if one can choose the best protocol more frequently, which is the case for small characterization intervals. For larger characterization intervals, the hybrid protocol becomes less attractive, and sometimes is completely ineffective. For small characterization intervals, the improvement decreases for small and for large data blocks. The Write-back and the Update protocols provide the best performance at these extremes. The greatest improvement for the dynamic hybrid protocol occurs when the difference between the Write-back and Update is small.

The improvement gained using the dynamic hybrid protocol for MP3D is low (under 25%) for 64-byte and 128-byte blocks. The results of a trace-driven simulation [7] for MP3D show that the performance of the page-based competitive algorithm [7] is the same as that of the Update protocol. The competitive algorithm can dynamically choose between write update and write invalidate. In the original simulation, the block size was a page [7]. This corresponds to our results for the 4K-byte or 8K-byte block sizes. Our results show that Update is better than the other basic protocols, and that the upper limit for improvement with the dynamic hybrid protocol is less than 15% at these block sizes. Our analytical prediction also agrees with the simulation results given in [1]. The upper limits for improvement for the dynamic hybrid protocol for LU (30%) and Barnes (60%) are larger than for MP3D.

The improvements for applications from the SPLASH-2 [6] benchmark suite, as predicted by our extended model, are given in Table 8, while Appendix B contains the data access characterization, the performance prediction and graphs depicting the relative improvement in performance for the four characteristic applications: Cholesky, Ocean (contiguous), Ocean (noncontiguous), and FFT. The results were obtained by assuming a 16-processor bus-based multiprocessor (see Table 4). The access pattern characterization of these applications shows a high percentage of accesses of SRSW type. For eight out of 12 applications the SRSW pattern is dominant for all block and characterization interval sizes. For the other four applications (LU - noncontiguous, FFT, Ocean - contiguous, and Ocean - noncontiguous) the SRSW pattern is dominant either for blocks smaller than 256 bytes or for characterization intervals smaller than 10^5 processor cycles. Since there is a high percentage of SRSW accesses, the Write-back protocol is the best choice for nine of these applications, even for large blocks (1K bytes and more). For the other three applications (LU - noncontiguous, FFT, and Ocean - noncontiguous), the best choice is the Update protocol when using blocks larger than 1K bytes.

It is apparent that only a few applications show a significant performance improvement with the dynamic hybrid protocol when blocks are smaller than 256 bytes. For two applications the improvement is greater than 50%, and for seven applications it is greater than 30%. Although patterns of data sharing were not examined in [6], a breakdown of miss rates for the SPLASH-2 applications was given. Based on this breakdown, certain conclusions about how blocks are accessed can be made. The results presented for Radix, Barnes, and FMM show a large percentage of misses due to both true and false sharing, indicating that besides sequential sharing of data blocks, there is also a significant amount of concurrent sharing. The Update protocol is more suitable for concurrent sharing while the Write-back protocol is better for sequential sharing (SRSW data access pattern). As a result, these three applications are good candidates for performance improvement using the dynamic hybrid protocol, and the results in Table 8 substantiate this. With blocks larger than 1K bytes, 11 applications have significantly improved performance (more than 50%) with the dynamic hybrid protocol. The frequency with which protocols are changed has a larger effect when small block sizes are used than when larger block sizes are used.

6: Conclusions

We have presented a set of analytical models for predicting the performance of parallel applications under various cache coherence protocol assumptions. Each of our models is applicable to several different system environments, including tightly-coupled shared-memory multiprocessors and implementations of distributed shared memory on workstation clusters. The models differ in the number and types of parameters they require. The simplest, *core* model, uses parameters that can easily be provided by a compiler. The more sophisticated models extend the core model to take the λ parameter, the η parameter, and transients into account. They require parameters that are best obtained from analyzing address traces generated by simulators or from monitoring previous runs. Our models are unique in that they lie between the many theoretical models and the many address-trace oriented models that have been proposed in literature.

We assessed the accuracy of our models by comparing their predictions with the results of simulation runs. We were able to show that on representative 15 applications even our simple core model was able to accurately predict the *relative* performance; i.e. which protocol performed better than the others. Hence, this model is suitable for compile time usage to implement *hybrid* protocols that allow different data blocks to use different coherence protocols, and *dynamic hybrid* protocols that also allow for changes in the choice of protocol during the run time of the application. As a result, we believe that with the availability of this model, dynamic hybrid protocols can be implemented with no specialized hardware support for run-time data access monitoring and decision making. The extended models provide for more accurate prediction of *absolute* performance. We were able to show that the models can predict the performance to within 10% of actual (simulated) executions.

Our study also provided us with other interesting insights. For example, we characterized the memory access behavior of several applications, representing the characterization in visual form. We also assessed the potential advantage of dynamic hybrid protocols and showed that the advantages are limited, especially for smaller block sizes, under the assumption that protocols support a sequential consistency model as in this paper. Recent research has shown that dynamic hybrid protocols can be more effective under other consistency models. For example, under release consistency most coherence traffic occurs at synchronization points and it is possible to coalesce multiple writes to the same data block in the local caches without requiring coherence traffic between synchronization points. This significantly changes the communication patterns, and Dahlgren recently showed that dynamic hybrid protocols are much more effective under these circumstances [34]. Our models can be easily adjusted to take these different traffic patterns into account.

In our current work, we are assessing how the working set size of applications, the number of processors, different consistency models, and different communication networks might affect the performance of dynamic hybrid protocols. In particular, we are about to assess the accuracy of our models in predicting the performance of distributed shared memory systems that run on workstation clusters. Finally, we intend to incorporate the core model into a compiler that can insert instructions into the code capable of managing a dynamic hybrid protocol. This code will then be run on a simulated implementation of the University of Toronto NUMAchine multiprocessor [4], where we will be able to accurately measure the improvement.

Acknowledgments

The authors wish to thank D. Vrsalovic, K. Sevcik, T. Abdelrahman, T. Mowry, N. Elezovic, A. Grbic, R. Grindley, V. Sruk, H. Bunjevac, T. Grcanac, J. Radej, G. Omrcen-Ceko, and A. Budin for their valuable comments on the research reported in this paper.

References

[1] J.E. Veenstra and R.J. Fowler, “A Performance Evaluation of Optimal Hybrid Cache Coherency Protocols”, In *Proceedings of Fifth International Conference on Architectural Support for Languages and Operating Systems* ASPLOS-V, Boston, Massachusetts, pages 149-160, October 1992.

[2] V. Balasundaram, “A Mechanism for Keeping Useful Internal Information in Parallel Programing Tools: The Data Access Descriptor”, *Journal of Parallel and Distributed Computing,* (9), pages 154-169, June 1990.

[3] F. Mounes-Toussi and D.J. Lilja, “The Potential of Compile-Time Analysis to Adapt the Cache Coherence Enforcement Strategy to the Data Sharing Characteristics”, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 6, No. 5, pages 470-481, May 1995.

[4] Z.G. Vranesic *et al.*, “The NUMAchine Multiprocessor”, *Technical Report CSRI-324*, Computer Systems Research Institute, University of Toronto, Toronto, Canada, 1995.

[5] J.P. Singh, W.-D. Weber, and A. Gupta, “SPLASH: Stanford Parallel Applications for Shared Memory”, *Computer Architecture News,* 20(1), pages 5-44, March 1992.

[6] S.C. Woo *et al.*, “The SPLASH-2 Programs: Characterization and Methodological Consideration”, In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pages 24-36, June 1995.

[7] A.W. Wilson, R.P. LaRowe, and M.J. Teller, “Hardware Assist for Distributed Shared Memory”, In *Proceedings of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, pages 246-255, May 1993.

[8] J.K. Bennett, J.B. Carter, and W. Zwaenepoel, “Adaptive Software Cache Management for Distributed Shared Memory Architecture”, In *International Conference on Supercomputing*, Seattle, Washington, pages 125-134, May 1990.

[9] A. Duda, “Analysis of Multicast-based Object Replication Strategies in Distributed Systems”, In *Proceedings of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, pages 311-318, May 1993.

[10] M. Stumm and S. Zhou, “Algorithms Implementing Distributed Shared Memory”, *IEEE Computer*, Vol 23, No. 5, pages 54-64, May 1990.

[11] J.B. Carter, J.K. Bennett, and W. Zwaenepoel, “Techniques for Reducing Consistency-Related Communication in Distributed Systems”, to appear in *ACM Transactions on Computer Systems*

[12] M. Dubois and F.A. Briggs, “Effects of Cache Coherency in Multiprocessors”, *IEEE Transactions on Computers*, Vol. c-31, No. 11, pages 1083-1099, November 1982.

[13] J.H. Patel, “Analysis of Multiprocessors with Private Cache Memories”, *IEEE Transactions on Computers*, Vol. c-31, No. 4, pages 296-304, November 1982.

[14] Q. Yang, L.N. Bhuyan, and B.-C. Liu, “Analysis and Comparison of cache Coherence Protocols for a Packet-Switch Multiprocessor”, *IEEE Transactions on Computers*, Vol. 38, pages 1143-1153, August 1988.

[15] A. R. Karlin *et al.*, “Competitive Snoopy Caching”, In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 244-254, October 1986.

[16] Z. Li, F. Mounes-Toussi, and D.J. Lilja, “Using Complier Assistance to Reduce the Network Traffic Requirements of a Directory-Based Cache Coherence Mechanism”, *Technical Report No: HPPC-95-01*, University of Minnesota, January 1995.

[17] S.J. Eggers, “Simplicity Versus Accuracy in a Model of Cache Coherency Overhead”, *IEEE Transactions on Computers*, Vol. 40, No. 8, pages 893-906, August 1991.

[18] S.J. Eggers and R.H. Katz, “A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation”, In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pages 257-270, May 1988.

[19] M. Dubois and J.-C. Wang, “Shared Block Contention in a Cache Coherence Protocol”, *IEEE Transactions on Computers*, Vol. 40, No. 5, pages 640-644, May 1991.

[20] M. Dubois and J.-C. Wang, “Shared Data Contention in a Cache Coherence Protocol”, In *Proceedings of the 1988 International Conference on Parallel Processing*, Vol. I, pages 146-155, August 1988.

[21] J. Archibald and J.-L. Baer, “Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model”, *ACM Transaction on Computer Systems*, Vol. 4, No. 4, pages 273-298, November 1986.

[22] J. Archibald, “A Cache Coherence Approach for Large Multiprocessor Systems”, In *International Conference on Supercomputing*, St. Malo, France, pages 337-345, July 1988.

[23] A.L. Cox and R.J. Fowler, “Adaptive Cache Coherency for Detecting Migratory Shared Data”, In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, San Diego, California, pages 98-108, May 1993.

[24] P. Stenstrom, M. Brorsson, and L. Sandberg, “An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing”, In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, San Diego, California, pages 109-118, May 1993.

[25] M. Brorsson and P. Stenstrom, “Visualizing Sharing Behavior in Relation to Shared Memory Management”, In *Proceedings of 1992 International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, pages 528-536, December 1992.

[26] M. Brorsson and P. Stenstrom, “Modelling Accesses to Migratory and Producer-Consumer Characterized Data in a Shared-Memory Multiprocessor”, In *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, pages 612-619, October 1994.

[27] M. Brorsson, “SM-prof: A Tool to Visualize and Find Cache Coherence Performance Bottlenecks in Multiprocessor programs”, In *Proceedings of the 1995 ACM SIGMETRICS and Perfromance’95, International Conference on Measurement & Modeling of Computer Systems*, Ottawa, Canada, pages 178-187, May 1995.

[28] S. Srbljic, “Model of Distributed Processing in Flexible Manufacturing Systems”, *Ph.D.* dissertation, Institute for Electronics, Faculty of Electrical Engineering, University of Zagreb, Croatia, November 1990. (Work published in Croatian, original title: “Model distribuirane obrade u prilagodljivim proizvodnim sustavima”)

[29] S. Srbljic and L. Budin, “Analytical Performance Evaluation of Data Replication Based Shared Memory Model”, In *Proceedings of the Second IEEE International Symposium on High Performance Distributed Computing*, Spokane, Washington, pages 326-335, July 1993.

[30] S. Srbljic, Z.G. Vranesic, and L. Budin, “Performance Prediction for Different Consistency Schemes in Distributed Shared Memory Systems”, In *Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing*, San Francisco, California, pages 295-302, August 1994.

[31] J. Heinrich, “MIPS R4000 User’s Manual”, Prentice-Hall, Inc., 1993.

[32] MIPS Technologies Inc., “Mips Open RISC Technology R10000”, *Microprocessor Technical Brief*, October 1994. (http://www.mips.com/HTMLs/r10000_docs/r10000_Pr_Info/R10000_Tech_Br_cv.html)

[33] J.E. Veenstra, “Mint Tutorial and User Manual”, *Technical Report 452*, Computer Science Department, University of Rochester, May 1993.

[34] F. Dahlgren, “Boosting the Performance of Hybrid Snooping Cache Protocols”, In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pages 60-69, June 1995.

Appendix A

Tables in this Appendix show the probabilities π_k for system events E_k for the MRSW and SRMW data access patterns. System events E_k are defined in Table 1 and the data access patterns and their parameters β, ρ, σ, and ξ are defined in Table 2. The interleaving parameters λ and η are defined in Section 4.1 and 4.6, respectively. The formulas are derived from the steady-state analysis of the system with infinite caches; therefore, the probabilities of events π_2 and π_{11} for the Update protocol and π_{12} for the Write-back are equal to zero. The interleaving parameters λ and η do not affect the performance of the Update protocol and uncached operations.

Table 9: **Probabilities for system events for the MRSW pattern**

Write-through	**Write-back**
$\pi_2 = \beta \rho \sigma / (\rho + \sigma)$	$\pi_2 = \beta \rho \sigma / (\rho + \sigma) - \beta \rho \sigma / (\rho + \beta \sigma)$
$\pi_8 = \rho$	$\pi_3 = \beta \rho \sigma / (\rho + \beta \sigma)$
	$\pi_5 = \beta \rho \sigma / (\rho + \beta \sigma)$
Update	**Uncached**
$\pi_{10} = \rho$	$\pi_1 = 1 - \rho$ $\pi_4 = \rho$

Table 10: **Probabilities for system events for the SRMW pattern**

Write-through	**Write-back**
$\pi_2 = (1 - \rho - \beta \xi) \beta \xi$	$\pi_3 = (1 - \rho - \beta \xi) \beta \xi$
$\pi_8 = (1 - \beta \xi) \rho + \beta \xi^2 / (\rho + \beta \xi)$	$\pi_5 = (1 - \rho - \beta \xi) \beta \xi^2 / (\rho + \beta \xi) + (1 - \rho - \beta \xi) \beta \rho \xi / (\rho + \beta \xi)$
$\pi_9 = \beta \rho \xi + (\beta \rho \xi + \beta (\beta - 1) \xi^2) / (\rho + \beta \xi)$	$\pi_6 = \beta (\beta - 1) \xi^2 (1 - \rho - \beta \xi) / (\rho + \beta \xi)$
	$\pi_7 = \beta \rho \xi + \beta \rho \xi / (\rho + \beta \xi) + \beta (\beta - 1) \xi^2$
Update	**Uncached**
$\pi_{10} = \rho + \beta \xi$	$\pi_1 = 1 - \rho - \beta \xi$ $\pi_4 = \rho + \beta \xi$

Table 11: **Probabilities for system events for the MRSW pattern with the λ parameter**

Write-through	**Write-back**
$\pi_2 = \beta \rho \sigma / (\rho + (\lambda + 1) \sigma)$	$\pi_2 = \beta \rho \sigma / (\rho + (\lambda + 1) \sigma) - \beta \rho \sigma / (\rho + (\lambda + 1) \beta \sigma)$
$\pi_8 = \rho$	$\pi_3 = \beta \rho \sigma / (\rho + (\lambda + 1) \beta \sigma)$
	$\pi_5 = \beta \rho \sigma / ((\lambda + 1) \rho + \beta \sigma)$

Table 12: Probabilities for system events for the SRMW pattern with the λ parameter

Write-through
$\pi_2 = (1-\rho-\beta\xi)\beta\xi / (\beta\xi (\lambda+1)(1-\beta\xi))$
$\pi_8 = (\lambda+1)(1-\beta\xi)\rho / (\beta \xi+(\lambda+1)(1-\beta\xi)) + (\lambda+1)\beta\xi^2 / (\rho+(\lambda+\beta)\xi)$
$\pi_9 = \beta\rho\xi / (\beta\xi+(\lambda+1)(1-\beta\xi)) + (\beta\rho\xi+\beta(\beta-1)\xi^2) / (\rho+(\lambda+\beta)\xi)$
Write-back
$\pi_3 = (1-\rho-\beta\xi)\beta\xi / (\beta\xi (\lambda+1)(1-\beta\xi))$
$\pi_5 = (\lambda+1)(1-\rho-\beta\xi)\beta\xi^2 / (\lambda\xi+1)(\rho+(\lambda+\beta)\xi) + (\lambda+1)(1-\rho-\beta\xi)\beta\rho\xi / ((\beta\xi+(\lambda+1)(1-\beta\xi))((\lambda+1)\rho+\beta\xi))$
$\pi_6 = \beta(\beta-1)\xi^2(1-\rho-\beta\xi) / ((\lambda\xi+1)(\rho+(\lambda+\beta)\xi))$
$\pi_7 = \beta\rho\xi / (\beta\xi+(\lambda+1)(1-\beta\xi)) + \beta\rho\xi / (\rho+(\lambda+\beta)\xi) + \beta(\beta-1)\xi^2 / (\lambda\xi+1)$

Table 13: Probabilities for system events for the MRSW pattern with the η parameter

Write-through	Write-back
$\pi_2 = (\eta + 1) \beta \rho \sigma / ((\eta + 1) \rho + \sigma)$	$\pi_2 = (\eta + 1) \beta \rho \sigma / ((\eta + 1) \rho + \sigma) - \beta \rho \sigma / ((\eta + 1) \rho + \beta \sigma)$
$\pi_8 = \rho$	$\pi_3 = \beta \rho \sigma / ((\eta + 1) \rho + \beta \sigma)$
	$\pi_5 = (\eta + 1) \beta \rho \sigma / (\rho + (\eta + 1) \beta \sigma)$

Table 14: Probabilities for system events for the SRMW pattern with the η parameter

Write-through
$\pi_2 = (\eta+1)(1-\rho-\beta\xi)\beta\xi / (\eta\beta\xi+1)$
$\pi_8 = (1-\beta\xi)\rho / (\eta\beta\xi+1) + \beta\xi^2 / ((\eta+1)\rho+\beta\xi)$
$\pi_9 = (\eta+1)\beta\rho\xi / (\eta\beta\xi+1) + ((\eta+1)\beta\rho\xi+\beta(\beta-1)\xi^2) / ((\eta+1)\rho+\beta\xi)$
Write-back
$\pi_3 = (\eta+1)(1-\rho-\beta\xi)\beta\xi) / (\eta\beta\xi+1)$
$\pi_5 = (\eta+1)(1-\rho-\beta\xi)\beta\xi^2 / ((\eta(1-\beta\xi)+1)((\eta+1)\rho+\beta\xi)) + (\eta+1)(1-\rho-\beta\xi)\beta\rho\xi / ((\eta\beta\xi+1)(\rho+(\eta+1)\beta\xi))$
$\pi_6 = (\eta+1)\beta(\beta-1)\xi^2(1-\rho-\beta\xi) / ((\eta(1-\beta\xi)+1)((\eta+1)\rho+\beta\xi))$
$\pi_7 = (\eta+1)\beta\rho\xi / (\eta\beta\xi+1) + (\eta+1)\beta\rho\xi / ((\eta+1)\rho+\beta\xi) + \beta(\beta-1)\xi^2 / (\eta(1-\beta\xi)+1)$

Appendix B

The figures in this Appendix show the data access pattern characterization, analytical prediction, and percent improvement for the dynamic hybrid protocol for four applications from the SPLASH-2 benchmark suite [23] using a 16-processor system (see Table 4). We assume that the dynamic hybrid protocol can choose the best basic protocol each 10^3 processor cycles.

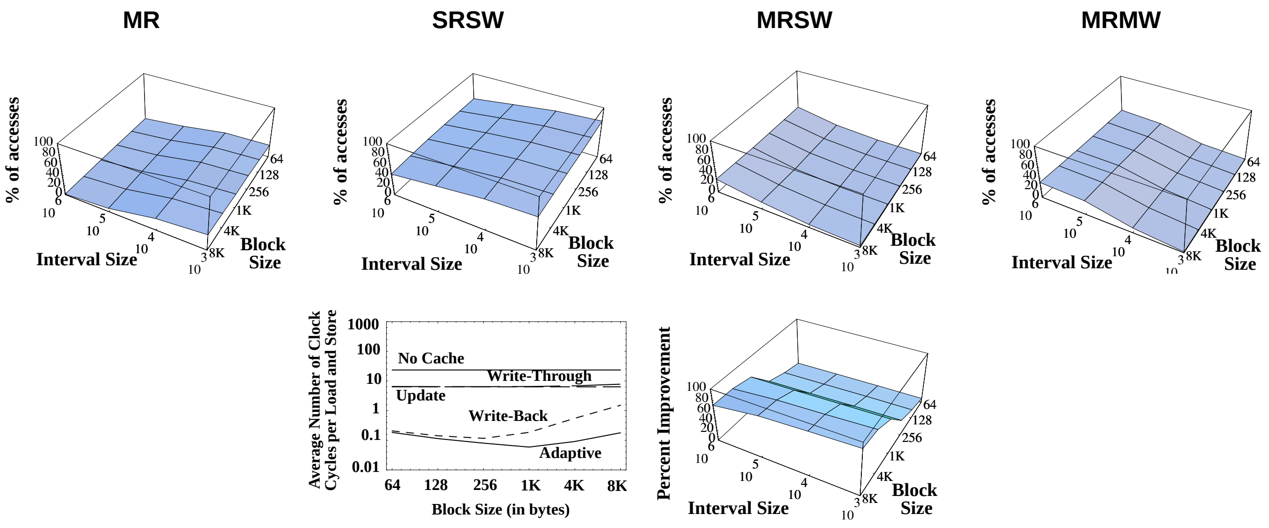


Figure 5: Sparse Cholesky Factorization
(File: tk.14.0, Postpass partition size: 32, 16384 byte cache)

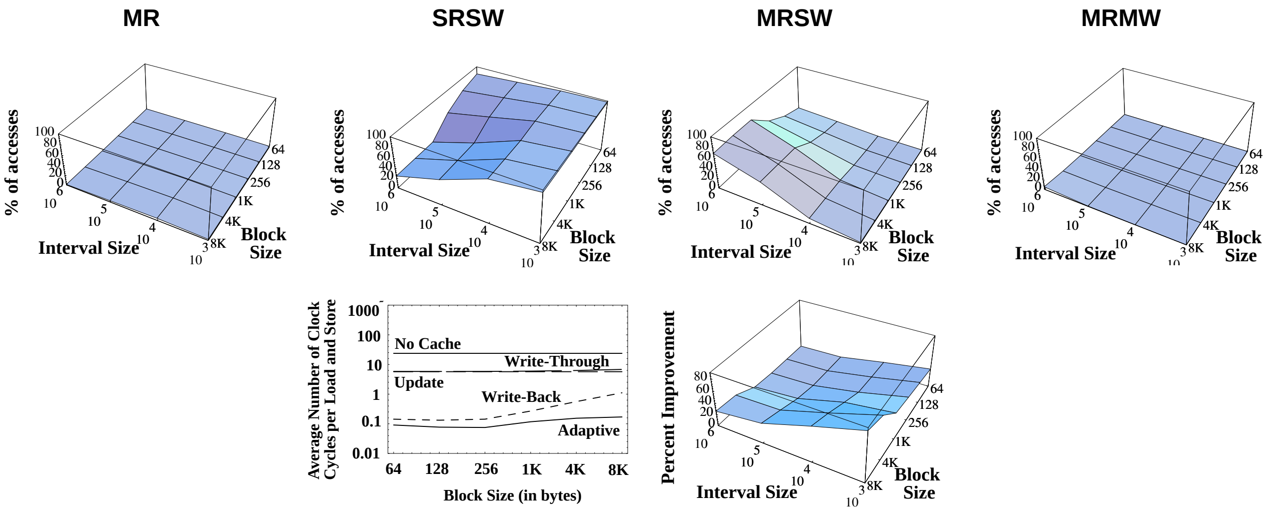


Figure 6: Ocean simulation with W-cycle multigrid solver (contiguous)
(Grid size = 258 x 258, Grid resolution (meters) = 20000.00, Time between relaxations (seconds) = 28800, Error tolerance = 1e-07)

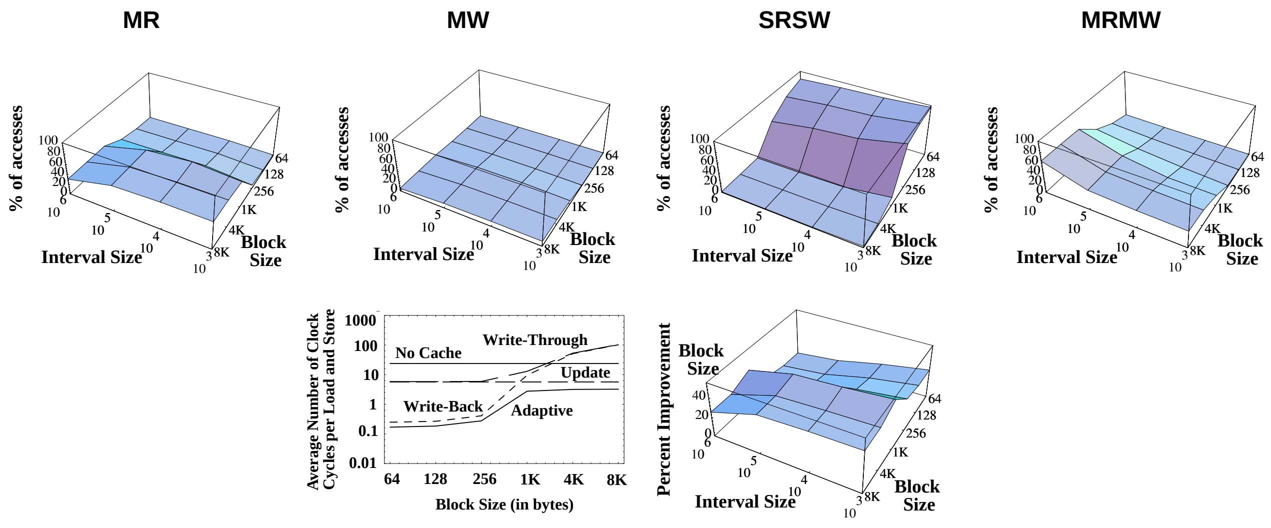


Figure 7: **Ocean simulation with W-cycle multigrid solver (noncontiguous)**
(Grid size = 258 x 258, Grid resolution (meters) = 20000.00, Time between relaxations (seconds) = 28800, Error tolerance = 1e-07)

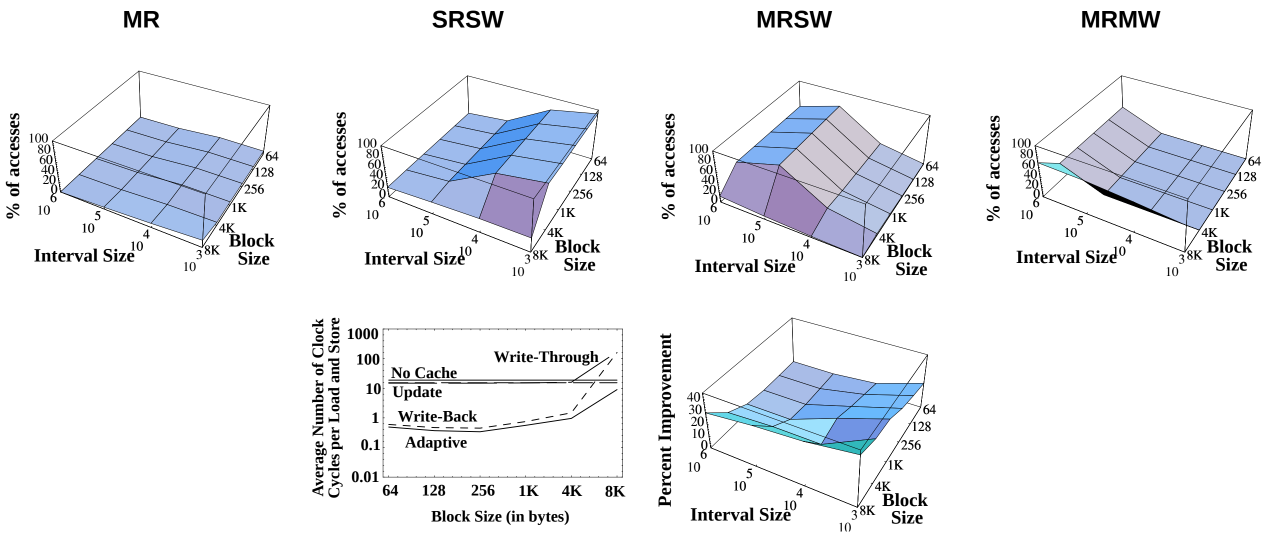


Figure 8: **FFT with Blocking Transpose**
(Total complex data points transformed: 2^{12}, 1024 Complex Doubles, 65536 Cache lines, 16 Byte line size, 4096 Bytes per page)