Analytical Prediction of Performance for Cache Coherence Protocols

Sinisa Srbljic, Zvonko G. Vranesic, *Senior Member*, *IEEE*
Michael Stumm, *Member*, *IEEE Computer Society*, and Leo Budin, *Member*, *IEEE*

Abstract—In this paper, we introduce new analytical models for predicting the performance of parallel applications under various cache coherence protocol assumptions. The purpose of these models is to determine which protocols are to be used for which data blocks, and, in the case of dynamic protocols, also to determine when to change protocols. Although we focus on tightly-coupled multiprocessor systems, similar models can be derived for 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 enable us to choose different input parameters depending on how the models will be used and depending on the needed accuracy in performance prediction.

We present the models and show how the required parameters can be obtained. We assess the accuracy of our models on 15 parallel applications. For these applications, our most complete model predicts performance within a 10 percent margin when compared to a simulation of a sequentially consistent multiprocessor system. As part of this study, we also show the potential advantage of using dynamic hybrid protocols.

Index Terms—Cache coherence, distributed shared memory, memory access behavior, analytical performance prediction, performance evaluation, dynamic hybrid protocols.

1 Introduction

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. 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. In the case of hybrid protocols, these models are intended to be used to determine which protocols to use for which data blocks, and, in the case of dynamic protocols, to determine when to change protocols.

Each model in the set 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 each type of access occurring. The core model serves two purposes in this paper. First, it is a useful model in its own right; while it is not very accurate in predicting the absolute performance, it is very useful in predicting the relative performance of the different cache coherence protocols. Second, it serves as a convenient vehicle for explaining the derivation of more complex models that predict absolute performance more accurately. These more complex models require as input the interleaving parameters that characterize and describe the ordering of accesses performed by different processors on each data block. These parameters might be estimated using modern compiler technology, but the parameters can probably be obtained more accurately by analyzing address traces generated by simulations or by monitoring previous runs (assuming that monitoring hardware is available to allow nonintrusive run-time profiling [2], [3]).

The accuracy and usefulness of our models is assessed by comparing the performance predicted by the models with the results of simulated execution for 15 parallel applications, mostly from the SPLASH and SPLASH-2 benchmark suites [4], [5], and also with simulation results reported by others [1], [6]. These comparisons show that our core model is capable of choosing the right cache coherence protocol for all the applications we considered, and the predictions of our more sophisticated models lie within 10 percent of the simulation results. As a side effect of our studies, we are also able to show the benefits of supporting hybrid and dynamic hybrid protocols, and that the benefits of dynamic protocols are limited to some applications. Although we focus on tightly-coupled multiprocessor systems

- *S. Srbljic and L. Budin are with the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia.*
  E-mail: {sinisa, leo}@zemris.fer.hr.
- *Z.G. Vranesic and M. Stumm are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 3G4. E-mail: {zvonko, stumm}@eecg.toronto.edu.*

Manuscript received 24 Aug. 1995; revised 10 June 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105633.

in this paper, we believe that similar models can also be derived for loosely-coupled distributed memory systems [6], [7], [8], [9].

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 [8], [10], [11], [12]. While simple, these models are too inaccurate in their predictions to be useful for our purposes. The second group of models is mostly *experimental* in nature, in that the models use information from complete memory traces of real applications [2], [13], [14], [15], [16], [17], or fully simulate the applications [1], [7], [18], [19], [20], [21]. 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 models is theoretical in that it makes no assumptions about the interleaving of memory accesses from different processors. Instead of assuming that accesses to shared data are uniformly distributed across all processors, we characterize the access patterns to blocks of data according to the number of processors performing accesses and the type of accesses they perform. For each of these patterns, we define additional parameters such as the probabilities of various types of 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 that add parameters to 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 the interleaving parameters. Dynamic hybrid protocols are considered in Section 5.

2 The Core Model

For 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 study [16], [17]. As is typical for theoretical models [8], [10], [11], [12], we also assume that accesses performed by different processors are independent in time, making no assumptions on the interleaving of these accesses. However, we do not assume that accesses are uniformly distributed across all processors, as is usually done in theoretical models. We introduce a few predetermined data access patterns according to the number of processors which perform accesses and according to the type of accesses. We note that Dubois and Wang introduced the burst parameters assuming that accesses are not independent in time; but, they assumed that the access bursts are independent in time and are uniformly distributed across a subset of processors [16], [17].

The concept of classifying data by how it is accessed (i.e., degree of sharing and access mode) has been used by many researchers for various purposes. For example, Weber and Gupta [22] proposed several classes of data objects that can be distinguished by their use in parallel programs and by their invalidation traffic patterns. Carter et al. [7] used the concept of data classification for coherence protocol selection in the Munin DSM system. Brorsson and Stenstrom [23], [24], [25] visualized the statistics of accesses to different data classes in order to analyze performance of applications running on the systems with limited-directory write-invalidate cache coherence protocol. Adve et al. [26] compared hardware and software cache coherence protocols for each introduced data class. We propose a classification that enables efficient comparison of different cache coherence protocols (Write-back, Write-through, Update, and uncached accesses). While most researches have tried to find the data classes that can be accurately distinguished by their features [7], [22], [23], [24], [25], we define additional parameters that facilitate quantitative comparison of different coherence protocols for each data class [27], [28], [29]. Adve et al. [26] also introduced additional parameters.

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 (Dirty copy) from remote cache and also write back to memory
E_4	Read one word from local cache (cache hit)
Store Instruction	
E_5	Write one word to memory
E_6	Obtain ownership and invalidate other copies
E_7	Read a data block from memory and invalidate other copies
E_8	Read a data block (Dirty copy) from remote cache
E_9	Write one word to local cache (hit to Dirty copy)
E_{10}	Update the memory and invalidate all other copies
E_{11}	Update the memory, invalidate all other copies, and read a data block from memory [1]
E_{12}	Update both the memory and all caches
E_{13}	Update the memory and all caches and read a data block from memory
Ejection of Dirty copy	
E_{14}	Write back a data block to memory

1. This event occurs when a write is issued to a cache in Invalid state; both ownership and data block must be obtained.

2. Invalid, Valid, and Dirty states are defined in the standard way for the Write-Back/Invalidate protocols.

However, they chose the data classes and parameters to efficiently compare performance of hardware and software coherence protocols, while we optimize our choice of data classes and parameters to compare different coherence protocols.

Our choice of access patterns and parameters, as well as how they can be obtained, is similar to the Brorsson and Stenstrom's workload model [23], [24], [25], although both models were developed independently. While we develop analytical expressions in order to directly predict the performance based on the workload parameters, Brorsson and Stenstrom use their workload parameters as input to a memory reference generator. They show that, for an appropriate choice of time interval, for which the data access characterization is performed, the stream of memory references produced by the memory reference generator will generate the same cache miss ratio as the stream of memory references from a real application.

Section 3 will show that our choice of access patterns, their parameters, and appropriate choice of characterization interval, results in a sufficiently good prediction of relative performance. Hence, our core model can be used as a decision making tool to choose the best basic protocol without determining the exact interleaving of accesses performed by different processors.

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 four system events: E_2, E_4, E_{10}, and E_{11}. Each write access updates the memory and the copy in the local cache; all other copies are invalidated. The Write-back protocol can incur eight possible events: E_2, E_3, E_4, E_6, E_7, E_8, E_9, and E_{14}. 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 four events: E_2, E_4, E_{12}, and E_{13}. 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_5).

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). 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.

For each access pattern, we introduce parameters, such as the number of processors that perform accesses and the probabilities of each type of access. 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 that all β writers write with probability ξ, respectively.[1] Of course,

1. One might consider combining patterns MRSW, SRMW, and MRMW. This could be done in two different ways. One possibility is to assume that MRMW pattern substitutes MRSW and SRMW patterns. However, we found that this significantly reduces the accuracy of the model. Another possibility is to add new parameters for the MRMW pattern (for example, separate parameters that denote the number of processors that perform only reads, writes,...), but this would make the model mathematically intractable.

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 $1-\rho$	probability that the access is a write probability that the access is a read
MRSW	Multiple Reader Single Writer	β σ ρ $1-\rho-\beta\sigma$	number of multiple readers probability that the access is a read from one of the β multiple readers [2] probability that the access is a write from a single writer probability that the access is a read from a single writer
SRMW	Single Reader Multiple Writer	β ξ ρ $1-\rho-\beta\xi$	number of multiple writers probability that the access is a write from one of the β multiple writers [2] probability that the access is a write from a single reader probability that the access is a read from a single reader
MRMW	Multiple Reader Multiple Writer	β ρ/β $(1-\rho)/\beta$	number of processors that perform reads and writes probability that the access is a write from one of the β processors [2] 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 SRSW with ρ set to 0 or 1, respectively.

2. To simplify the derivation of expressions, it is assumed that all β processors perform these accesses with equal probability.

while these assumptions do not correspond to reality, they keep the analytical expressions simple, while still giving sufficiently good predictions (as we will show in Section 4.3). 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, in the case of MRMW, the set of processors that read from a data block will not be the same as the set of processors that write to the same block.

2.3 Cost Calculation for Different Data Access Patterns

The core 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)[2]:

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

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 for a given access pattern 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 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.

To illustrate this, we will derive the probabilities of events E_8, E_7, and E_3 for the Write-back protocol and MRMW pattern. System event E_8 is an exclusive read from a remote cache. The event occurs if a write from processor j follows a write from a different processor i (with 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 a read from processor i. The probability of sequences of the form $W_iR_iR_i, \ldots, R_iW_j$ is then

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

where ρ/β is 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 the equivalent of multiplying (2) by $\beta(\beta-1)$ and calculating the sum of the given series, we obtain the expression for π_8 given in Table 3.

System event E_7 occurs when a write is issued to a data block of which there are multiple copies in other caches but not in the local cache. The processor must obtain the copy

2. The system we model is relatively simple. For example, in evaluating the cost for the Write-through protocol, each write operation results in a memory transaction with an attendant cost. It is possible to assume more sophisticated hardware, say, with buffers capable of coalescing multiple writes operations to the same data block. This would significantly alter the cost for basic operations, and, hence, a new set of cost expressions would have to be developed, or the same set of expressions could be used but the data access characterization must be performed on the output of coalescing buffers.

TABLE 3
PROBABILITIES FOR SYSTEM EVENTS FOR THE MRMW PATTERN

Write-through	Write-back
$\pi_2 = \rho(\beta-1)(1-\rho)/(1+(\beta-1)\rho)$ $\pi_4 = (1-\rho)/(1+(\beta-1)\rho)$ $\pi_{10} = \rho-(\beta-1)\rho^2/(1+(\beta-1)\rho)$ $\pi_{11} = (\beta-1)\rho^2/(1+(\beta-1)\rho)$	$\pi_2 = \rho(\beta-1)(1-\rho)/(1+(\beta-1)\rho) - \rho(\beta-1)(1-\rho)/(\rho+\beta-1)$ $\pi_3 = \rho(\beta-1)(1-\rho)/(\rho+\beta-1)$ $\pi_4 = (1-\rho)/(1+(\beta-1)\rho)$ $\pi_6 = \rho-(\beta-1)\rho^2/(1+(\beta-1)\rho) - \rho^2/(\rho+\beta-1)$ $\pi_7 = (\beta-1)\rho^2/(1+(\beta-1)\rho) - (\beta-1)\rho^2/(\rho+\beta-1)$ $\pi_8 = (\beta-1)\rho^2/(\rho+\beta-1)$ $\pi_9 = \rho^2/(\rho+\beta-1)$
Update	**Uncached**
$\pi_4 = 1-\rho$ $\pi_{12} = \rho$	$\pi_1 = 1-\rho$ $\pi_5 = \rho$

1. The formulas are derived from the steady-state analysis of the system with infinite caches; therefore, the probabilities of events π_2 and π_{13} for the Update protocol and π_{14} for the Write-back protocol are equal to zero.
2. Since the probabilities of the system events related to the execution of the read operation should sum to the probability that access is a read, and, since the probabilities of events related to the execution of the write operation should sum to the probability that access is a write, the probabilities of the system events E_4 and E_9 can be derived based on the probabilities of other system events and will not be presented in the rest of the paper. They are given in this table only to show the completeness of the proposed model.
3. π_k is the probability of system event E_k defined in Table 1.

of the data block from memory and other copies in the system must be invalidated. To calculate probability π_7, we observe that event E_7 occurs on processor j when it issues a write W_j after the data block has been read by another processor k that is different from processor j and that is also different from processor i that issued the previous write W_i; i.e., if we denote the relation "happened before" with $<$, then $\exists\, k$: $W_i < R_k < W_j$, $i \neq k \neq j$. We first calculate the probability of the sequences $W_iQQ, \ldots, QW_j$ occurring, where Qs are reads from any processor except j.[3] The probability of Q is equal to $(\beta-1)(1-\rho)/\beta$.[4] We take into account the sequences for all possible pairs of processors by multiplying the probability of sequences $W_iQQ, \ldots, QW_j$ by $\beta(\beta-1)$. Finally, we subtract the probability of those sequences in which all reads are from processor i,[5] which is, in fact, equal to probability π_8 calculated above. Hence,

$$\pi_7 = \beta(\beta-1)\left(p\left(\sum_{z=0}^{\infty} W_i\underbrace{(Q, \ldots, Q)}_{z \text{ times}}W_j\right)\right) - \pi_8$$
$$= \beta(\beta-1)\sum_{z=0}^{\infty}\left(\frac{\rho}{\beta}\right)^2\left(\frac{(1-\rho)(\beta-1)}{\beta}\right)^z - \pi_8. \quad (3)$$

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

$$\pi_3 = \beta(\beta-1)\left(p\left(\sum_{z=0}^{\infty} W_i(\underbrace{R_i, \ldots, R_i}_{z \text{ times}})R_j\right)\right)$$
$$= \beta(\beta-1)\sum_{z=0}^{\infty}\frac{\rho}{\beta}\left(\frac{1-\rho}{\beta}\right)^{z+1}. \quad (4)$$

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 $\rho = 0$, $\rho = 1$, and $\beta = 1$, respectively. If we assume zero cost for hits in the local cache (system events E_4 and E_9), then all MR events that incur cost have probability zero for all cache coherence protocols, because, in the steady state, all accessing caches will have valid copies. Similarly, all SRSW events that incur overhead using the Write-back protocol also have probability zero, because, in steady state, the accessing cache will own the data block and have a local copy in the Dirty state, so all reads and writes remain local. Therefore, the average steady-state costs $C_{MR,*}$ and $C_{SRSW,WB}$ are equal to zero. The expressions for the MRSW and SRMW patterns are given in Appendix A.[6]

Each memory access results in exactly one of the given system events. Therefore, the sum of the probabilities of all system events for a given data access pattern and for a particular coherence protocol must be equal to one. It is easy to verify that this is actually the case for the probabilities for the MRMW pattern given in Table 3.

2.4 Predicting the Performance of Applications

Our core model partitions the data space into blocks of fixed size and then evaluates the cost of accessing each data block separately. For each data block, it is necessary to determine the access pattern and parameters and, then, to calculate the average steady-state cost per access, $C_{pattern,protocol}$. This value

3. If the read is from processor j, then system event E_7 would not be triggered, but, rather, system event E_6, which does not require a copy from memory.
4. The probability of Q can be calculated as the sum of the probabilities of reads from all processors except processor j, which is the probability of a read from one processor, $(1-\rho)/\beta$, multiplied by $(\beta-1)$.
5. If all reads Q in a sequence are from processor i, then the write W_j from processor j will not result in event E_7, but, rather, in event E_8, which is why we must subtract the probability π_8 to obtain π_7.

6. Appendix A actually shows the expressions for an extended model, but the expressions for the core model can be derived from them by setting λ and η to zero. Parameters λ and η are discussed in Section 4.

is then multiplied by the percentage of accesses to this block, and the corresponding values for all blocks are summed to obtain the average cost per shared access for a given application and cache coherence protocol:

$$C_{protocol}^{application} = \sum_{\text{block } i} \frac{\#\,\text{Block}_i}{\#\,\text{Appl}} C_{pattern(i),protocol} \tag{5}$$

where $C_{pattern(i),protocol}$ is the cost for an access to block i, whose data accesses are of type *pattern*(i), for the given coherence protocol. The terms $\#Block_i$ and $\#Appl$ denote the number of accesses to 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. For that, it is sufficient to just evaluate $C_{pattern(i),protocol}$ for each data block and for each coherence protocol being considered. We compute $C_{protocol}^{application}$ only to assess the quality of our model.

2.5 Access Pattern Characterization Using Smaller Time Intervals

Analysis of the data access patterns of various applications shows that the access pattern characterization of data blocks typically changes over time [25]. 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. The accuracy of the model can be improved by considering the access patterns for smaller time intervals. If the application run time is partitioned into smaller intervals, then the performance of the application can be predicted as:

$$C_{protocol}^{application} = \sum_{\substack{\text{block } i \\ \text{time interval } j}} \frac{\#\,\text{Block}_{ij}}{\#\,\text{Appl}} C_{pattern(i,j),protocol} \tag{6}$$

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 good results with our core 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 model for the cases considered in this paper. 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, we expect that advances in compiler technology will allow the estimation of the required parameters at compile time. The compiler might split the application into a group of statements or regions [30] along natural boundaries (for example synchronization points),[7] and, then, apply advanced data dependence analysis of the type required for parallelizing compilers [31] for each region to estimate the access pattern and the associated parameters. An example of such data dependence analysis can be found in [2]. This analysis is part of the compiler marking algorithm. In order to choose the best coherence protocol for each write access, the compiler marking algorithm must determine the exact interleaving of accesses performed by different processors. We believe that the parameters required for our model can be estimated more easily, because it is not necessary to predict the exact ordering among all accesses, but rather just the number of processors that perform the accesses and the probabilities of such 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 [4] and SPLASH-2 [5] benchmark suites. In this section, we present the results obtained from three applications. The first two, BARNES (512 particles) and MP3D (25,000 molecules, five steps, test.geom), are from SPLASH, and the third is LU decomposition (100×100 matrix). In Appendix B, we present results for SPLASH-2 applications.

The conclusion that can be drawn from the results presented here is that the core model is adequate for predicting the relative performance of the coherence protocols and can therefore be used to choose an appropriate basic protocol for each data block. Section 4 describes extensions to the core model that improve the accuracy, so that the extended model can be used to predict *absolute* performance.

3.1 Simulation Details

In order to obtain results for different architectural configurations, we simulated two bus-based multiprocessor systems: one having eight processors and the other having 16. To obtain realistic cost estimates, we simulated the systems with MIPS R4400 processor [33].[8] One 64-bit word can be transferred in one clock cycle in the eight-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,

7. While we use the characterization intervals of equal size for the cases presented in this paper, the characterization intervals need not be of the same size.

8. The parameters of the simulations were chosen to correspond to the NUMAchine multiprocessor prototype [3], which uses the R4400.

TABLE 4
SYSTEM EVENTS COSTS

8-Processor Multiprocessor [1] **64-Bit Data Bus**	**16-Processor Multiprocessor** [2] **128-Bit Data Bus**
Load Instruction	
$c_1 = 12$	$c_1 = 27$
$c_2 = 10 + h/8$	$c_2 = 26 + h/16$
$c_3 = 15 + h/8$	$c_3 = 29 + h/16$
$c_4 = 0$	$c_4 = 0$
Store Instruction	
$c_5 = 5$	$c_5 = 10$
$c_6 = 20$	$c_6 = 30$
$c_7 = 22 + h/8$	$c_7 = 32 + h/16$
$c_8 = 15 + h/8$	$c_8 = 29 + h/16$
$c_9 = 0$	$c_9 = 0$
$c_{10} = 20$	$c_{10} = 30$
$c_{11} = 22 + h/8$	$c_{11} = 32 + h/16$
$c_{12} = 20$	$c_{12} = 30$
$c_{13} = 22 + h/8$	$c_{13} = 32 + h/16$
Ejection of Dirty Copy	
$c_{14} = 4 + h/8$	$c_{14} = 10 + h/16$

1. The constants are determined based on the technical specifications of the MIPS R4400 processor [33].

2. The constants are determined based on simulation results of the University of Toronto NUMAchine multiprocessor [3], also using MIPS R4400 processor.

3. System events c_4 and c_9 are hits in the local cache and we assume zero cost for these events.

4. c_k is the cost of system event E_k denoting processor stall time in clock cycles; h is the block size in bytes.

memory and cache latency, and processor response time.[9] 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 eight-processor system. Section 5 will present results for the 16-processor system.

3.2 Data Access Pattern Characterization of Applications

Fig. 1 shows the data access pattern characterizations[10] of three applications: BARNES, MP3D, and LU, as obtained by processing address traces generated by the MINT program [34]. We give the characterization of shared data accesses for various data block and time interval sizes.[11] 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. While the graphs in this section are used to show general trends, Appendix B presents the values of selected points for the SPLASH-2 applications we considered in numerical form.

Each access type (i.e., MR, MW, SRSW, MRSW, SRMW, and MRMW) 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 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 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 data accesses will be of SRSW type.

Fig. 1a depicts the access characterization for BARNES. We can see that, even with a large block and characterization interval sizes, almost all of the shared data accesses are to MR blocks. 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 blocks become mostly SR blocks, which is included with the SRSW blocks.

9. Our model does not take bus and memory contention into account, but uses the same constants for overhead regardless of traffic load. Because of this, the prediction of our models will be inaccurate under heavy loads. However, in our experience, bus and memory contention only increase the difference between the coherence protocols, so the models are still useful for predicting the relative performance of the protocols. If the models are being used for predicting absolute performance, then different sets of constants would have to be used for different traffic loads. An alternative, perhaps more attractive, approach would be to use our models to predict the amount of traffic produced per load and store (which is independent of memory and bus contention) and then use Boothe and Ranade's Squeeze Model [35] to predict the latency on the basis of the estimated traffic. Our models can be used to predict network traffic in a relatively straightforward way by modifying the constants in Table 4 to represent the number of bits that need to be transferred instead of latency.

10. Note that we do not differentiate between true sharing (where different processors access the same shared data) and false sharing (where unrelated data accessed by different processors happens to be collocated in the same cache line). Both kinds of sharing have the same effect on performance.

11. Brorsson and Stenstrom use similar graphs to visualize the data access pattern characterization [23].

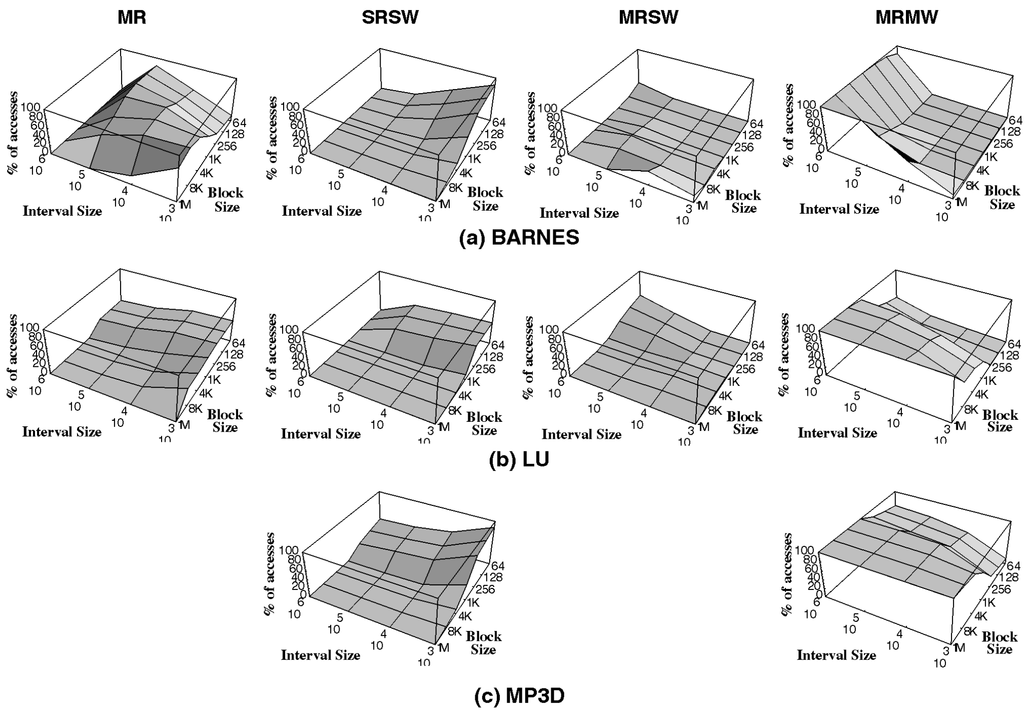


Fig. 1. Data access pattern characterization (interval size in processor cycles; block size in bytes): (a) BARNES, (b), LU, (c) MP3D.

Fig. 1b 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 Fig. 1c. It shows that there are only two significant types of shared data accesses for this application: MRMW and SRSW. As we decrease the characterization interval size, the percentage of MRMW type accesses decreases, while the percentage of SRSW accesses increases.

3.3 Simulation Results

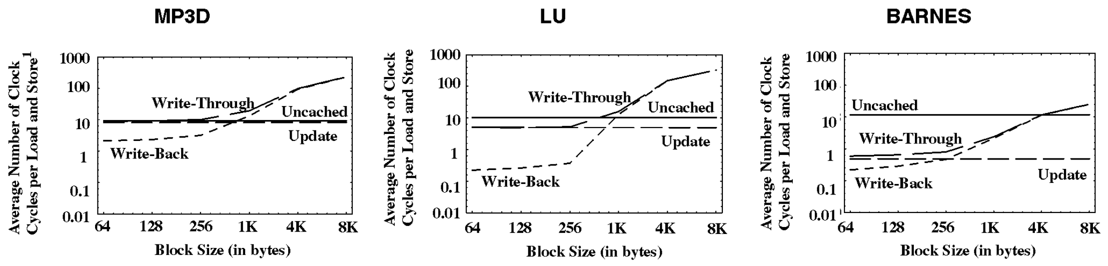
Fig. 2 shows the cost per access as predicted by our model and as determined by simulation runs for several cache coherence protocols. The first row shows the results of simulation, while the other two rows show the prediction of our model using the characterization interval sizes of 10^3 and 10^5 processor cycles, respectively. 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).

The leftmost graph in Fig. 2a shows the performance for MP3D obtained by simulation. For small blocks, the Write-back protocol outperforms the other protocols because most of the accesses are of SRSW type (see Fig. 1c). For SRSW accesses with the Write-back protocol, only the first write to a block incurs a cost, while each write to the block 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 (about 40 percent), and all accesses that are not of SRSW type are of MRMW type, the difference between Update, Write-through, and uncached operations is relatively 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.[12] 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 Fig. 1c). Therefore, Write-back and Write-through protocols give poor performance for large blocks.

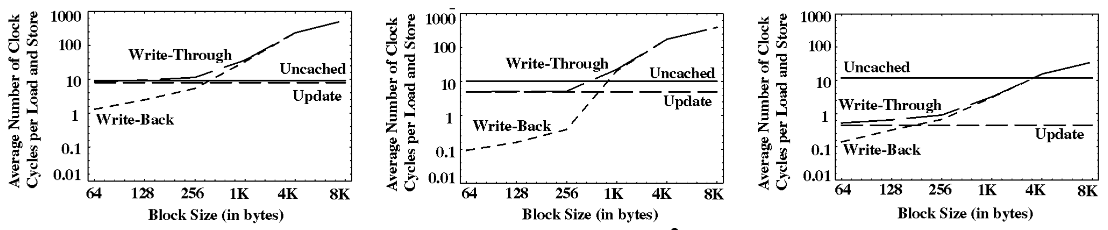
Fig. 2a also shows the simulated performance for LU decomposition. As in MP3D, the Write-back protocol performs best for small blocks. The cache hit rate is higher for LU than for MP3D, because data accesses that are not of SRSW type are of MR type, and because of the lower percentage of writes (about 24 percent). 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 Fig. 2a shows the simulated performance for BARNES. The difference between the performance for cached and uncached accesses is large because of the low percentage of writes (about 2 percent), and because most accesses are of SRSW type (see Fig. 1a). The Write-back protocol outperforms the other basic protocols for small blocks, while the Update protocol outperforms the other protocols for large blocks. It is interesting to note that processor manufactures now primarily support the Write-back protocol.

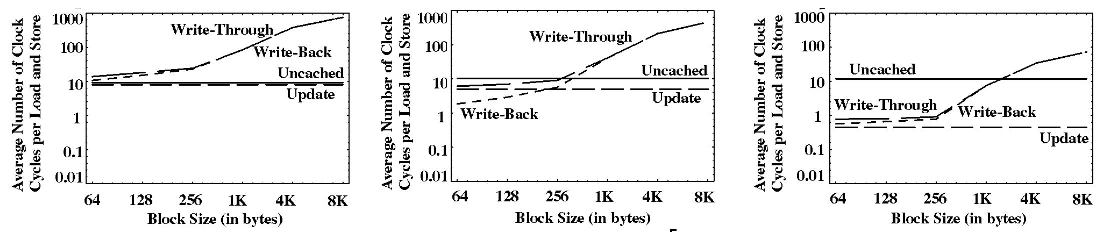
12. Performance of the Update protocol depends only on the read / write ratio and the cost of updating.



(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

Fig. 2. Comparison of analytical results as predicted by the core model and simulation results (eight-processor system) The number of load and store instructions is the same for all protocols for a given application. (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.

3.4 Appropriate Choice for Characterization Interval Size

The quality of results generated by the core model depends on the size of the characterization intervals (which are assumed to be equisized here[13]) if either Write-through or Write-back protocol is used. This is evident in Fig. 2, where the results with interval size set to 10^3 processor cycles (Fig. 2b) correspond more closely to those results produced by the simulator (Fig. 2a) than the results generated when the interval size is set to 10^5 (Fig. 2c).[14]

If the characterization interval is chosen 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 by the same processor is much lower than the probability of two accesses by different processors. In practice, however, accesses to blocks tend to come in bursts from a single processor at a time [14], [15], [16], [17]. This can also be seen in Fig. 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 nonzero if at least one of the processors performs a write). 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 ownership 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 is not critical for large data blocks. For large blocks, long bursts from a single processor are less likely, so data accesses behave, for the most part, as if they were independent in time. Moreover, the effects of transients are lower, because the average number of accesses per block is higher. For large data blocks, Fig. 2 confirms that the core model correctly predicts the relative performance for both interval sizes.

13. Our choice of equal sized characterization intervals stems from the fact that we have assumed only the availability of address traces. The model could be extended to include variable sized intervals, based on information that could be obtained from source code.

14. The size of the characterization interval does not affect the results if the applications run uncached or if the Update protocol is used, because the results depend only on the probability of writes, as can be seen from Table 3.

For smaller data blocks (≤ 256 bytes), the choice of characterization interval is much more critical. The interval must be chosen to minimize the effects of both access bursts and transients. Comparing Fig. 2a and Fig. 2c shows that even relative performance is not correctly predicted for large characterization intervals and small data blocks. Access bursts argue for using smaller characterization intervals, while transients argue for using larger characterization intervals. The appropriate size of characterization interval can, for example, be determined experimentally by comparing the simulation results with the predicted performance of the model for different characterization intervals. Alternatively, we have obtained good results in practice by choosing the size of the characterization interval to contain $2n^2/(n-1)$ accesses per accesses block, where n is the number of processors. This expression was derived by minimizing an error function that embodies the effects of two worst-case scenarios (that the core model does not account for), one for access bursts and one for transients.[15]

Our analysis of the given applications shows that relative performance can be correctly predicted if the characterization interval is chosen to be 10^3 processor cycles. In this case, there will be, on average, between 20 and 100 accesses per block. We have experimented with multiprocessor systems from eight to 64 processors, where the number of accesses should be between 19 and 130 accesses (for $n = 8$ and $n = 64$) using the above rule, and have obtained good results.

4 Extending the Core Model

This section describes extensions to the core model that improve the prediction of absolute performance, regardless of the size of the characterization interval. 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, so the extensions assume that some information on this ordering is available. The order of accesses does not affect the performance of uncached operations or the performance of the Update protocol, nor does it affect the performance of SRSW and MR data blocks with the write-invalidate protocols.

We consider two separate cases, depending on whether there is less or more interleaving of accesses by different processors than that assumed by the core model. In the core model, it is assumed that all processors have equal probability of accessing the data; that is, if $P(p_k)$ is the probability of processor p_k being the next processor to access the data, the model assumes that $P(p_i)/P(p_j) = 1$ for all i and j. To improve our prediction for the case where interleaving is less than the amount assumed by the core model, we introduce a parameter $\lambda \geq 0$. In this case, after processor p_i accesses the data, $P(p_i)/P(p_j)$ will be greater than one for all $j <> i$, and we set $\lambda = P(p_i)/P(p_j) - 1 \geq 0$. To predict the performance for the case where interleaving is greater than the amount assumed by the core model, i.e., $P(p_j)/P(p_i) > 1$ for all $j <> i$ after an access by p_i, we introduce a parameter $\eta = P(p_j)/P(p_i) - 1 \geq 0$.

We discuss both how to extend the expressions of the core model to include the λ and η parameters, as well as how to estimate the values of these two parameters. The two experimental models, Dubois and Wang's burst model [16], [17] and Eggers and Katz' write-run model [14], [15], also include interleaving parameters. We tried to estimate the parameters λ and η based on both the burst size [16], [17] and on the length of write run [14], [15]. Since the values of λ and η estimated by these two methods did not significantly improve the accuracy of our extended model, we introduce a new method based on the expected number of successive accesses that do not incur coherence overhead (as described below). While the interleaving parameters constitute the basic parameters in the models of [14], [15], [16], [17], the interleaving parameters in our model are used as auxiliary parameters that are used only to improve the accuracy in predicting the absolute values of performance.

We will show that the accuracy of the model is retained when the same *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

We redefine the probability π_k of each event E_k occurring by including λ. 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 $\rho(\lambda + 1)/(\lambda + \beta)$, the probability of a read by the same processor i will be $(1-\rho)(\lambda + 1)/(\lambda + \beta)$, the probability of a write by another processor will be $\rho/(\lambda + \beta)$, and the probability of a read by another processor will be $(1-\rho)/(\lambda + \beta)$. This insight allows us to redefine the probabilities of Table 3. For example, the probability of event E_8 for the MRMW pattern using the Write-back protocol becomes:

$$\pi_8 = \beta(\beta-1)\sum_{z=0}^{\infty}\frac{\rho}{\beta}\left(\frac{(1-\rho)(\lambda+1)}{\lambda+\beta}\right)^z\frac{\rho}{\lambda+\beta}. \qquad (7)$$

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 $W_iR_iR_i, \ldots, R_iW_j$ 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 $\lambda + 1$ times higher then the probability of a subsequent access by another processor.

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 the MR, MW, and SRSW patterns can again be derived from the expressions for MRMW by setting $\rho = 0$, $\rho = 1$, or $\beta = 1$. The expressions for the MRSW and SRMW patterns are given in Appendix A.[16]

15. The worst case for transients occurs during MR pattern, when all n processors read a data block from memory or remote caches. For all cached accesses (Write-back, Write-through, and Update) and MR pattern, the core model would predict a zero cost. The worst case for access bursts occurs when all accesses from one processor come in one burst. For write-invalidate protocols (Write-back and Write-through) and MW access pattern, the core model calculates a higher cost because it assumes that, after a write from a particular processor, all n processors have equal chance to perform the next write.

16. Again, the appropriate expressions can be obtained by setting η to zero in the expressions given in the appendix.

TABLE 5
PROBABILITIES FOR SYSTEM EVENTS FOR THE MRMW PATTERN WITH THE λ PARAMETER

Write-through	Write-back
$\pi_2 = \rho(\beta-1)(1-\rho)/(\lambda\rho+1+(\beta-1)\rho)$	$\pi_2 = \rho(\beta-1)(1-\rho)/(\lambda\rho+1+(\beta-1)\rho) - \rho(\beta-1)(1-\rho)/((\lambda+1)\rho+\beta-1)$
$\pi_{10} = \rho - (\beta-1)\rho^2/(\lambda\rho+1+(\beta-1)\rho)$	$\pi_3 = \rho(\beta-1)(1-\rho)/((\lambda+1)\rho+\beta-1)$
$\pi_{11} = (\beta-1)\rho^2/(\lambda\rho+1+(\beta-1)\rho)$	$\pi_6 = \rho - (\beta-1)\rho^2/(\lambda\rho+1+(\beta-1)\rho) - (\lambda+1)\rho^2/((\lambda+1)\rho+\beta-1)$
	$\pi_7 = (\beta-1)\rho^2/(\lambda\rho+1+(\beta-1)\rho) - (\beta-1)\rho^2/((\lambda+1)\rho+\beta-1)$
	$\pi_8 = (\beta-1)\rho^2/((\lambda+1)\rho+\beta-1)$

Π_k is the probability of system event E_k defined in Table 1.

4.2 Estimating λ from a Memory Trace

In order to estimate λ, we measure or estimate the average number of successive local accesses in the characterization intervals, where an access is considered local if it does 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.

One way to obtain the average numbers of successive local accesses to a block from a memory trace is to introduce an *n*-bit state variable in which each bit corresponds to one of the *n* processors. On each read, the bit corresponding to the accessing processor is set to one, 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 and almost insignificant relative to the cost of transferring a data block, 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_6, π_7, and π_8 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_6 is excluded from 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.

4.3 Accuracy of Our Models

Fig. 3 shows the difference between the simulation results and the predictions of our models for the Write-back protocol. There are four curves that show the importance of the λ parameter. Curve 4 shows how poor the accuracy of the prediction of the core model can be if an inappropriate characterization interval size is used. In this case, a large interval was chosen (10^5 processor cycles), and the difference between the predictions of the core model and the results of the simulation can be huge. In contrast, Curve 3 shows the accuracy of the prediction of the core model for the smaller, more appropriate, characterization interval size of 10^3 processor cycles. We found that, if we use this fixed interval size for the SPLASH and SPLASH-2 applications, the difference between the model and the simulation will usually be less than 100 percent (an order of magnitude smaller than in the case of the larger interval size).

Curve 1 shows the difference between the simulation results and the prediction of our extended model with the λ parameter when using the larger interval size of 10^5 processor cycles. The difference is quite small and, usually, less then 10 percent, even though an inappropriate characterization interval is being used. 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. Note, however, that, in the case of Curve 1, a separate set of parameter values was used for each block and each characterization interval.

Finally, Curve 2 considers a variation where the same averaged set of parameters is used for each characterization interval of each type. That is, the parameters are collected from the memory trace and averaged within each of the six access patterns of Table 2 before being used in the model (Table 6 lists the averaged parameters for the case of a 10^5-cycles characterization interval and 64-byte block size). It is interesting to observe that performance prediction using only a small number of averaged parameters can be as accurate as the predictions stemming from more accurate information driven by an entire memory trace. This characteristic makes it possible to use our models for large applications for which it is impractical to store the entire memory trace. It should be noted, however, that the performance prediction for dynamic hybrid protocols (discussed in

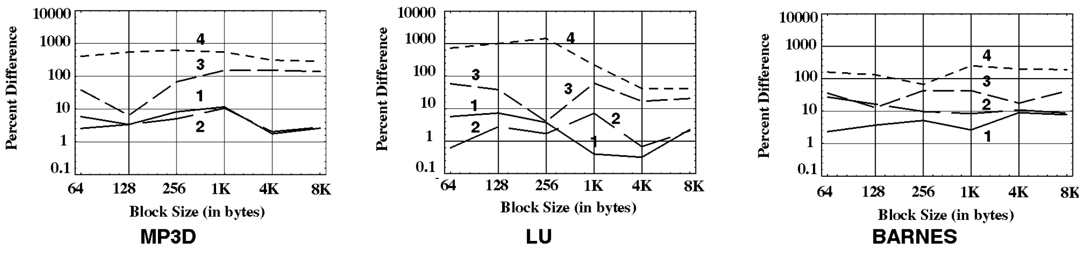


Fig. 3. Write-back protocol: Percent difference between simulation results and the analytical prediction (eight-processor system). 1) Performance prediction based on prediction for each characterization interval and for each block separately—λ parameter included, interval size = 10^5. 2) Performance prediction based on averaged parameters—λ parameter included (see Table 6), interval size = 10^5 (Write to Shared state is considered as local access for 8K and 4K byte blocks, otherwise is remote). 3) Interval size = 10^3, parameter $\lambda = 0$. 4) Interval size = 10^5, parameter $\lambda = 0$.

TABLE 6
AVERAGED PATTERN PARAMETERS

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	$\beta = 2.000$, $\lambda = 2.098$	NA		0.002	$\beta = 8.000$, $\lambda = 0$
SRSW	11.256	$\rho = 0.067$	40.659	$\rho = 0.499$	41.640	$\rho = 0.292$
MRSW	2.551	$\rho = 0.191$, $\sigma = 0.153$, $\beta = 2.678$, $\lambda = 7.575$	20.153	$\rho = 0.068$, $\sigma = 0.108$, $\beta = 6.961$, $\lambda = 145.524$	0.030	$\rho = 0.072$, $\sigma = 0.125$, $\beta = 5.537$, $\lambda = 1.718$
SRMW	0.100	$\rho = 0.431$, $\xi = 0.117$, $\beta = 1.000$, $\lambda = 0.501$	0.221	$\rho = 0.007$, $\xi = 0.077$, $\beta = 4.522$, $\lambda = 0.487$	NA	
MRMW	4.240	$\rho = 0.219$, $\beta = 6.026$, $\lambda = 59.398$	11.232	$\rho = 0.263$, $\beta = 4.780$, $\lambda = 84.513$	58.078	$\rho = 0.477$, $\beta = 6.489$, $\lambda = 43.254$

Eight-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 it is defined as a local access.

Section 5) cannot be done based on averaged parameters, but must be done with parameters for each block and characterization interval.

Fig. 3 shows the results for the Write-back protocol only, but our model behaves equally well for the other protocols.

4.4 Taking Transients into Account

While the λ parameter allows the use of large characterization intervals in our model, it is also necessary to take transient effects into account when operating with small characterization intervals (10^3 processor cycles or less). The transient cost is particularly important for blocks whose data accesses for a given characterization interval are of MR or SRSW type, because their cost is assumed to be zero, even though there is a cost associated with the initial loading of a block.

The transient cost can be estimated by introducing two n-bit state variables per data block, one for write-invalidate protocols and one for Update protocol, where n is the number of processors. Each bit of this variable corresponds to one processor to indicate if the cache has a valid copy of the data block. 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, and in that the remaining bits are not cleared for writes. The transient cost for a particular data access pattern is calculated by comparing the state of these variables and data access pattern with those of previous intervals. Based on the differences in the states and data access patterns, we can determine the number of system events which will incur cost.[17] The costs for these events, which are given in Table 4, are multiplied by the number of the events. The products are then summed and averaged over the number of accesses in a given characterization interval to calculate the average transient cost per access:

$$C_{transient(i,j),protocol} = \frac{\sum c_k m_k}{\# \text{Block}_{ij}} \tag{8}$$

where $C_{transient(i,j),protocol}$ is the average transient cost per access for data block i during the characterization interval j for the given protocol, c_k is the cost of system event E_k, m_k is the number of system events E_k, and #Block$_{ij}$ is the number of accesses to block i during interval j. The average transient cost per access is then added to the cost imposed by sharing during the characterization interval:

$$C_{i,j,protocol} = C_{pattern(i,j),protocol} + C_{transient(i,j),protocol} \tag{9}$$

4.5 Adding the η Parameter

Even after including the transient costs, there still remain some differences between the simulation results and the predictions of our model. The reason is that, for small characterization

17. For example, in the case where MR pattern follows the SRSW pattern, based on the difference in the state variable for write-invalidate protocol, we can determine the number of processors that read the data block from memory or remote cache.

TABLE 7
PROBABILITIES FOR SYSTEM EVENTS FOR THE MRMW PATTERN WITH THE η PARAMETER

Write-through
$\pi_2 = (\eta + 1)\rho(\beta - 1)(1 - \rho)/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho)$
$\pi_{10} = \rho - (\eta + 1)(\beta - 1)\rho^2/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho)$
$\pi_{11} = (\eta + 1)(\beta - 1)\rho^2/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho)$
Write-back
$\pi_2 = (\eta + 1)\rho(\beta - 1)(1 - \rho)/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho) - (\eta + 1)\rho(\beta - 1)(1 - \rho)/((\beta - 1)\eta + \beta + \rho - 1)$
$\pi_3 = (\eta + 1)\rho(\beta - 1)(1 - \rho)/((\beta - 1)\eta + \beta + \rho - 1)$
$\pi_6 = \rho - (\eta + 1)(\beta - 1)\rho^2/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho) - \rho^2/((\beta - 1)\eta + \beta + \rho - 1)$
$\pi_7 = (\eta + 1)(\beta - 1)\rho^2/(\rho(\beta - 1)\eta + (1 - \rho)\eta + 1 + (\beta - 1)\rho) - (\eta + 1)(\beta - 1)\rho^2/((\beta - 1)\eta + \beta + \rho - 1)$
$\pi_8 = (\eta + 1)(\beta - 1)\rho^2/((\beta - 1)\eta + \beta + \rho - 1)$

intervals and small blocks, interleaving is higher in practice for the MW, MRSW, SRMW, and MRMW accesses than assumed by our model. To take this discrepancy into account, we introduce a new parameter η. 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 being from the same processor is then defined to be η + 1 times lower than the probability of an access being from another processor. This parameter is incorporated into the core model similar to the way λ was added. 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. The expressions that include both the λ and η for the MRSW and SRMW patterns are given in Appendix A.

The expressions in Tables 5 and 7 are presented separately, so that the reader can get a better understanding of the effects of each parameter. Our model, however, combines both parameters (even though either λ or η will always be zero). With the addition of these two parameters, the difference between the results generated by simulation and our model is less then 10 percent 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. Different coherence protocols are used for different blocks and over different time intervals. In this section, we discuss the predicted performance of dynamic hybrid protocols for a variety of applications from the SPLASH [4] and SPLASH-2 [5] benchmark suites. In particular, we show how two parameters, the block size and the frequency at which the coherence protocol is chosen, affect the performance of the dynamic hybrid protocol.

Design, implementation, and analysis of dynamic hybrid protocols has received attention in previous research [2], [6], [7], [13], [18], [19], [20], [21], [36], [37], [38], [39], [40]. Both hardware [13], [18], [19] and software [7], [37] implementations have been proposed, as have those that combine hardware and software [6]. Inevitably, a function is used to determine which protocol to use when. This decision can be done based on run-time information [6], [13], [18], [19], compile-time information [7], or a combination of both [2].

To adapt our models for hybrid protocols, we rearrange (6) of Section 2.5. Instead of using the cost expression for a single basic protocol for all characterization intervals, we choose the cost expression of the basic protocol that for the characterization interval and block gives the lowest cost. This allows us to predict the performance of the dynamic hybrid protocol that chooses the best basic protocol for each characterization interval and block as:

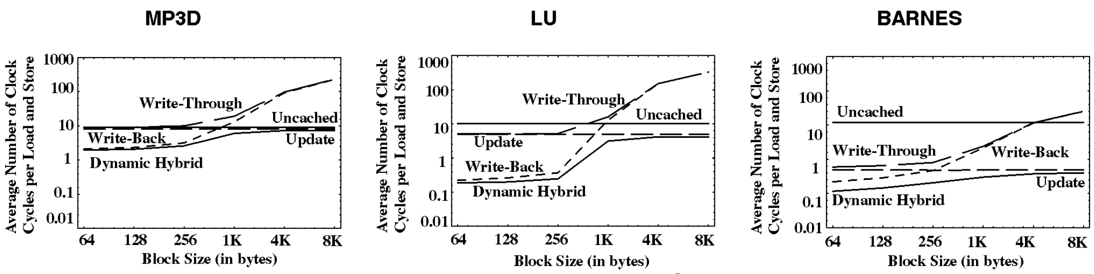
$$C_{hybrid} = \sum_{\substack{\text{block } i \\ \text{time interval } j}} \frac{\#\,\text{Block}_{ij}}{\#\,\text{Appl}}\, min_{protocol}\left(C_{i,j,protocol}\right). \qquad (10)$$

Fig. 4a 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 the performance of the Write-back protocol for small blocks, and better than the 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, so our results represent an upper bound on the improvement one might expect from a real system.

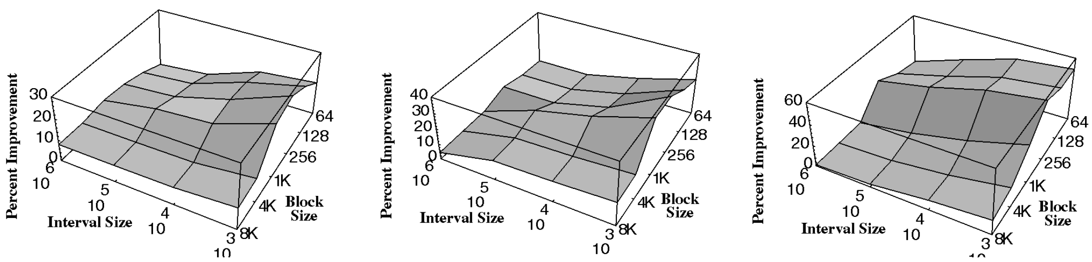
Fig. 4b shows the improvement for the dynamic hybrid protocol. We calculate the improvement attained with dynamic hybrid protocol as:

$$Improvement = 100 \times (C_{best} - C_{hybrid}) / (C_{best}), \qquad (11)$$

where $C_{best} = min_{protocol}(C_{protocol})$ is the cost of the basic protocol that gives the best result for a given data block and interval size. 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 can be 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.



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



(b) The percent improvement for dynamic hybrid protocol with respect to the best non-hybrid protocol

Fig. 4. Analytical prediction as predicted by the extended analytical model (λ and η parameters included, as well as transients) (eight-processor system). (a) Performance prediction (interval size = 10^3 processor cycles. (b) The percent improvement for dynamic hybrid protocol with respect to the best nonhybrid protocol.

The greatest improvement for the dynamic hybrid protocol occurs when the difference between the Write-back and Update is small.

The improvement obtained with the dynamic hybrid protocol for MP3D is low (under 25 percent) for 64-byte and 128-byte blocks. This result agrees with simulation results obtained by others. A simulation of MP3D by Wilson et al. shows that the performance of a page-based competitive algorithm is the same as that of the Update protocol [6]. (The competitive algorithm can dynamically choose between write update and write invalidate.) Wilson et al. used a block size of a page, which corresponds to our 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 percent for these block sizes. Our analytical predictions also agree with the results of simulation by Veenstra and Fowler [1]. The upper limits for improvement for the dynamic hybrid protocol for LU (30 percent) and Barnes (60 percent) are larger than for MP3D.

The improvements for other applications from the SPLASH-2 [5] benchmark suite, as predicted by our extended model, are given in Table 8.[18] 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 SRSW accesses. For seven of 12 applications, the SRSW pattern is prevalent for all block and characterization interval sizes (see Table 11 in Appendix B). For the other five applications (Ocean–contiguous, Ocean–noncontiguous, FFT, LU–contiguous, and LU–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 eight of these applications, even for large blocks (1K bytes and more). For the other four applications (Radix, Ocean–noncontiguous, FFT, and LU–noncontiguous), the best choice is the Update protocol when using blocks larger than 1K bytes (see Table 12 in Appendix B).

From these results, it is apparent that only 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 percent, and, for seven applications, it is greater than 30 percent. Although patterns of data sharing were not examined by Woo et al. in their study [5], 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 percent) 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.

18. A technical report [32] contains graphs depicting the data access characterization, the performance prediction and the relative improvement in performance for four of the SPLASH-2 applications: Cholesky, Ocean (contiguous), Ocean (noncontiguous), and FFT. Appendix B presents the data access characterization and performance prediction for twelve applications from the SPLASH-2 benchmark suite.

TABLE 8
THE PERCENT IMPROVEMENT FOR DYNAMIC HYBRID PROTOCOL WITH RESPECT TO THE BEST NONHYBRID PROTOCOL FOR SPLASH-2 APPLICATIONS AS PREDICTED BY THE EXTENDED ANALYTICAL MODEL

Application	Interval Size	Block size					
		64 bytes	128 bytes	256 bytes	1K bytes	4K bytes	8K bytes
Radix	10^3	53.4	66.7	73.1	85.3	93.2	93.0
	10^6	1.7	18.0	34.8	70.2	89.1	88.8
Barnes	10^3	50.1	53.3	60.5	77.5	89.3	93.8
	10^6	19.2	15.8	17.8	38.0	76.6	86.9
Water-Spatial	10^3	35.1	43.7	49.4	54.1	84.6	92.0
	10^6	27.7	27.0	21.3	10.1	76.7	89.8
FMM	10^3	33.7	36.7	41.3	56.5	85.3	91.6
	10^6	10.6	7.9	6.5	14.9	61.2	76.0
Ocean (contiguous)	10^3	31.9	36.3	40.7	45.5	75.3	85.7
	10^6	3.6	4.0	4.5	7.2	14.2	21.9
Raytrace	10^3	30.1	38.2	47.7	70.9	84.4	86.5
	10^6	27.3	35.7	45.1	66.7	80.0	82.2
Cholesky	10^3	8.1	17.0	29.9	67.7	84.5	88.9
	10^6	4.6	12.0	23.1	62.0	69.3	69.6
Water-Nsquared	10^3	11.7	15.3	24.2	41.8	76.4	86.0
	10^6	3.6	3.0	2.9	4.1	58.0	72.7
Ocean (noncontiguous)	10^3	29.4	29.8	33.4	51.3	43.9	43.0
	10^6	5.4	5.7	9.5	34.1	25.5	25.5
FFT	10^3	15.8	21.5	27.4	33.8	36.4	41.7
	10^6	2.4	3.7	5.3	8.1	12.6	29.7
LU (contiguous)	10^3	5.9	14.6	22.2	27.8	43.1	70.8
	10^6	0.9	1.4	2.6	5.6	25.4	31.1
LU (noncontiguous)	10^3	5.6	6.7	5.3	34.1	65.7	28.0
	10^6	1.4	2.3	0.0	10.6	60.6	13.7

λ and η parameters included, as well as transients; 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).

TABLE 9
PROBABILITIES FOR SYSTEM EVENTS FOR THE MRSW PATTERN

Write-through	Write-back
$\pi_2 = (\eta+1)\beta\rho\sigma/((\eta+1)\rho+(\lambda+1)\sigma)$	$\pi_2 = (\eta+1)\beta\rho\sigma/((\eta+1)\rho+(\lambda+1)\sigma) - \beta\rho\sigma/((\eta+1)\rho+(\lambda+1)\beta\sigma)$
$\pi_{10} = \rho$	$\pi_3 = \beta\rho\sigma/((\eta+1)\rho+(\lambda+1)\beta\sigma)$
	$\pi_6 = (\eta+1)\beta\rho\sigma/((\lambda+1)\rho+(\eta+1)\beta\sigma)$
Update	**Uncached**
$\pi_{12} = \rho$	$\pi_1 = 1-\rho$
	$\pi_5 = \rho$

6 CONCLUSIONS

We have presented a set of analytical models for predicting the performance of parallel applications under various cache coherence protocol assumptions. The models differ in the number and types of parameters they require and in the level of accuracy of performance prediction they provide. For the simplest, *core* model, we expect that advances in compiler technology will allow the estimation of the required parameters at compile time. The more sophisticated models extend the core model by adding the λ and η parameters, and taking transients into account. These parameters are best obtained by analyzing address traces generated from simulators or by 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 15 representative applications, even our simple core model was able to correctly predict the *relative* performance; i.e., which protocol performed better than the others. Hence, this model is suitable for compile time use 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 percent of actual (simulated) executions.

Our study also provided other interesting insights. For example, we characterized the memory access behavior of

TABLE 10
PROBABILITIES FOR SYSTEM EVENTS FOR THE SRMW PATTERN

Write-through	
$\pi_2 = (\eta + 1)(1 - \rho - \beta\xi)\beta\xi/((\eta\beta\xi + 1)(\beta\xi + (\lambda + 1)(1 - \beta\xi)))$	
$\pi_{10} = (\lambda + 1)(1 - \beta\xi)\rho/((\eta\beta\xi + 1)(\beta\xi + (\lambda + 1)(1 - \beta\xi))) + (\lambda + 1)\beta\xi^2/((\eta + 1)\rho + (\lambda + \beta)\xi)$	
$\pi_{11} = (\eta + 1)\beta\rho\xi/((\eta\beta\xi + 1)(\beta\xi + (\lambda + 1)(1 - \beta\xi))) + ((\eta + 1)\beta\rho\xi + \beta(\beta - 1)\xi^2)/((\eta + 1)\rho + (\lambda + \beta)\xi)$	
Write-back	
$\pi_3 = (\eta + 1)(1 - \rho - \beta\xi)\beta\xi)/((\eta\beta\xi + 1)(\beta\xi + (\lambda + 1)(1 - \beta\xi)))$	
$\pi_6 = (\lambda + 1)(\eta + 1)(1 - \rho - \beta\xi)\beta\xi^2/((\eta(1 - \beta\xi) + 1)((\eta + 1)\rho + (\lambda + \beta)\xi)(\lambda\xi + 1)) +$ $(\lambda + 1)(\eta + 1)(1 - \rho - \beta\xi)\beta\rho\xi/((\eta\beta\xi + 1)((\lambda + 1)\rho + (\eta + 1)\beta\xi)(\beta\xi + (\lambda + 1)(1 - \beta\xi)))$	
$\pi_7 = (\eta + 1)\beta(\beta - 1)\xi^2(1 - \rho - \beta\xi)/((\eta(1 - \beta\xi) + 1)((\eta + 1)\rho + (\lambda + \beta)\xi)(\lambda\xi + 1))$	
$\pi_8 = (\eta + 1)\beta\rho\xi/((\eta\beta\xi + 1)(\beta\xi + (\lambda + 1)(1 - \beta\xi))) + (\eta + 1)\beta\rho\xi/((\eta + 1)\rho + (\lambda + \beta)\xi) + \beta(\beta - 1)\xi^2/((\eta(1 - \beta\xi) + 1)(\lambda\xi + 1))$	
Update	**Uncached**
$\pi_{12} = \rho + \beta\xi$	$\pi_1 = 1 - \rho - \beta\xi$
	$\pi_5 = \rho + \beta\xi$

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. However, recent research has shown that dynamic hybrid protocols can be more effective if multiple writes to the same data block can be coalesced into a single transaction. This significantly changes the communication patterns, and Dahlgren recently showed that dynamic hybrid protocols are much more effective under these circumstances [36].

In our current work, we are assessing how the working set size of applications, the number of processors, and different communication networks might affect the performance of dynamic hybrid protocols. In particular, we will 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 University of Toronto NUMAchine multiprocessor [3], where we will be able to accurately measure the improvement.

APPENDIX A

Tables 9 and 10 show the probabilities, π_k, of system events E_k occurring for the MRSW and SRMW data access patterns. The 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 Sections 4.1 and 4.5, 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. We assume zero cost for events E_4 and E_9, so probabilities π_4 and π_9 are not presented in the tables. The interleaving parameters λ and η do not affect the performance of the Update protocol and uncached operations.

APPENDIX B

Table 11 shows the data access pattern characterization for applications from the SPLASH-2 benchmark suite [5], assuming a 16-processor system (see Table 4). Table 12 presents the performance as predicted by the extended analytical model for different cache coherence protocols, including the dynamic hybrid protocol.

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,” *Proc. Fifth Int’l Conf. Architectural Support for Languages and Operating Systems, ASPLOS-V*, pp. 149-160, Boston, Oct. 1992.

[2] 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 Trans. Parallel and Distributed Systems*, vol. 6, no. 5, pp. 470-481, May 1995.

[3] Z.G. Vranesic et al., “The NUMAchine Multiprocessor,” Technical Report CSRI-324, Computer Systems Research Inst., Univ. of Toronto, Canada, 1995.

[4] J.P. Singh, W.-D. Weber, and A. Gupta, “SPLASH: Stanford Parallel Applications for Shared Memory,” *Computer Architecture News*, vol. 20, no. 1, pp. 5-44, Mar. 1992.

[5] S.C. Woo et al., “The SPLASH-2 Programs: Characterization and Methodological Consideration,” *Proc. 22nd Ann. Int’l Symp. Computer Architecture*, pp. 24-36, Santa Margherita Ligure, Italy, June 1995.

[6] A.W. Wilson, R.P. LaRowe, and M.J. Teller, “Hardware Assist for Distributed Shared Memory,” *Proc. 13th Int’l Conf. Distributed Computing Systems*, pp. 246-255, Pittsburgh, Penn., May 1993.

[7] J.B. Carter, J.K. Bennett, and W. Zwaenepoel, “Techniques for Reducing Consistency-Related Communication in Distributed Shared-Memory Systems,” *ACM Trans. Computer Systems*, vol. 13, no. 3, pp. 205-243, Aug. 1995.

[8] A. Duda, “Analysis of Multicast-Based Object Replication Strategies in Distributed Systems,” *Proc. 13th Int’l Conf. Distributed Computing Systems*, pp. 311-318, Pittsburgh, Penn., May 1993.

[9] M. Stumm and S. Zhou, “Algorithms Implementing Distributed Shared Memory,” *Computer*, vol. 23, no. 5, pp. 54-64, May 1990.

[10] M. Dubois and F.A. Briggs, “Effects of Cache Coherency in Multiprocessors,” *IEEE Trans. Computers*, vol. 31, no. 11, pp. 1,083-1,099, Nov. 1982.

TABLE 11
THE PERCENTAGE OF DIFFERENT TYPES OF ACCESSES

Application	Interval Size	Block Size	Data Access Pattern (% of accesses)					
			MR	MW	SRSW	MRSW	SRMW	MRMW
Radix	10^6	8K	0.0	0.0	86.8	0.2	5.1	7.9
	10^3	8K	1.2	2.5	91.1	0.0	0.0	5.1
	10^6	64	0.2	0.0	87.8	7.8	3.8	0.5
	10^3	64	0.2	0.0	99.7	0.0	0.0	0.1
Barnes	10^6	8K	11.8	0.0	74.8	0.2	0.0	13.2
	10^3	8K	17.6	0.0	80.5	1.2	0.0	0.6
	10^6	64	14.2	0.0	75.3	7.6	0.0	2.9
	10^3	64	5.8	0.0	94.1	0.1	0.0	0.1
Water-Spatial	10^6	8K	7.6	0.0	72.7	1.3	0.0	18.3
	10^3	8K	15.3	0.0	81.4	1.9	0.0	1.3
	10^6	64	20.0	0.0	78.5	1.4	0.0	0.0
	10^3	64	0.5	0.0	99.4	0.0	0.0	0.0
FMM	10^6	8K	0.2	0.0	61.0	2.7	0.1	36.0
	10^3	8K	12.4	0.0	76.5	8.5	0.1	2.4
	10^6	64	17.6	0.0	69.1	9.9	0.1	3.2
	10^3	64	2.3	0.0	97.5	0.2	0.0	0.1
Ocean (contiguous)	10^6	8K	1.1	0.0	24.6	68.9	0.1	5.3
	10^3	8K	2.4	0.0	95.7	1.8	0.0	0.0
	10^6	64	2.6	0.0	85.4	11.8	0.0	0.0
	10^3	64	1.7	0.0	98.0	0.3	0.0	0.0
Raytrace	10^6	8K	27.9	0.0	71.4	0.1	0.0	0.6
	10^3	8K	21.6	0.0	78.2	0.0	0.0	0.2
	10^6	64	26.6	0.0	73.2	0.1	0.0	0.2
	10^3	64	7.8	0.0	92.0	0.0	0.0	0.1
Cholesky	10^6	8K	1.1	0.0	41.9	26.4	0.0	30.6
	10^3	8K	30.3	0.0	64.1	3.6	0.0	1.9
	10^6	64	2.8	0.0	47.0	22.7	0.0	27.5
	10^3	64	24.5	0.0	74.2	0.6	0.0	0.6
Water-Nsquared	10^6	8K	23.3	0.0	71.8	0.5	0.0	4.3
	10^3	8K	10.4	0.0	89.4	0.2	0.0	0.0
	10^6	64	26.4	0.0	72.5	0.8	0.0	0.3
	10^3	64	1.3	0.0	98.7	0.0	0.0	0.0
Ocean (noncontiguous)	10^6	8K	30.3	5.4	1.3	0.0	0.0	63.1
	10^3	8K	53.6	10.0	2.9	0.6	0.0	32.9
	10^6	64	3.7	0.6	84.5	4.4	0.1	6.7
	10^3	64	2.2	0.2	96.8	0.4	0.0	0.4
FFT	10^6	8K	0.0	0.0	19.5	12.2	0.0	68.3
	10^3	8K	14.4	5.0	28.1	0.1	0.0	52.2
	10^6	64	0.0	0.0	19.9	46.8	0.0	33.3
	10^3	64	6.4	0.1	92.7	0.8	0.0	0.1
LU (contiguous)	10^6	8K	0.1	0.0	29.0	57.1	0.0	13.7
	10^3	8K	27.8	0.0	65.9	6.3	0.0	0.0
	10^6	64	3.2	0.0	57.1	39.8	0.0	0.0
	10^3	64	27.3	0.0	72.6	0.0	0.0	0.0
LU (noncontiguous)	10^6	8K	5.4	0.0	8.3	0.0	0.0	86.3
	10^3	8K	27.7	0.0	15.9	0.8	0.2	55.5
	10^6	64	8.4	0.0	55.2	36.4	0.0	0.0
	10^3	64	27.4	0.0	72.6	0.0	0.0	0.0

Interval size in processor cycles; Block size in bytes.

[11] J.H. Patel, “Analysis of Multiprocessors with Private Cache Memories,” *IEEE Trans. Computers*, vol. 31, no. 4, pp. 296-304, Apr. 1982.

[12] Q. Yang, L.N. Bhuyan, and B.-C. Liu, “Analysis and Comparison of Cache Coherence Protocols for a Packet-Switch Multiprocessor,” *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,143-1,153, Aug. 1989.

[13] A.R. Karlin et al., “Competitive Snoopy Caching,” *Proc. 27th Ann. Symp. Foundations of Computer Science*, pp. 244-254, Oct. 1986.

[14] S.J. Eggers, “Simplicity Versus Accuracy in a Model of Cache Coherency Overhead,” *IEEE Trans. Computers*, vol. 40, no. 8, pp. 893-906, Aug. 1991.

[15] S.J. Eggers and R.H. Katz, “A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation,” *Proc. 15th Ann. Int’l Symp. Computer Architecture*, pp. 373-382, Honolulu, Haw., May 1988.

[16] M. Dubois and J.-C. Wang, “Shared Block Contention in a Cache Coherence Protocol,” *IEEE Trans. Computers*, vol. 40, no. 5, pp. 640-644, May 1991.

TABLE 12
AVERAGE NUMBER OF CLOCK CYCLES PER LOAD AND STORE PREDICTED BY THE EXTENDED ANALYTICAL MODEL

Application	Block Size	Cache coherence protocol				
		Uncached	Update	Write-through	Write-back	Hybrid
Radix	8K bytes	18.9	14.3	39.5	26.4	1.0
	64 bytes	18.9	14.4	14.5	0.2	0.1
Barnes	8K bytes	19.5	13.2	15.4	2.2	0.1
	64 bytes	19.5	13.3	13.3	0.1	0.05
Water-Spatial	8K bytes	21.6	9.5	11.6	2.2	0.2
	64 bytes	21.6	9.5	9.5	0.01	0.01
FMM	8K bytes	23.6	6.0	10.7	5.0	0.4
	64 bytes	23.6	6.0	6.0	0.1	0.1
Ocean (contiguous)	8K bytes	23.8	5.7	6.9	1.2	0.2
	64 bytes	23.8	5.7	5.8	0.1	0.1
Raytrace	8K bytes	22.2	8.5	8.7	0.3	0.04
	64 bytes	22.2	8.5	8.5	0.1	0.04
Cholesky	8K bytes	23.4	6.4	7.9	1.6	0.2
	64 bytes	23.4	6.5	6.5	0.2	0.2
Water-Nsquared	8K bytes	21.5	9.6	9.8	0.2	0.02
	64 bytes	21.5	9.7	9.7	0.02	0.02
Ocean (noncontiguous)	8K bytes	23.8	5.7	102.6	100.9	3.2
	64 bytes	23.8	5.7	5.8	0.2	0.2
FFT	8K bytes	18.8	15.4	140.6	133.0	9.0
	64 bytes	18.8	14.9	15.0	0.6	0.5
LU (contiguous)	8K bytes	21.2	10.1	12.3	2.3	0.7
	64 bytes	21.2	10.2	10.2	0.04	0.04
LU (noncontiguous)	8K bytes	21.2	10.2	57.8	50.0	7.3
	64 bytes	21.2	10.2	10.2	0.04	0.04

Interval size = 10^3 processor cycles; λ and η parameters included, as well as transients.

[17] M. Dubois and J.-C. Wang, “Shared Data Contention in a Cache Coherence Protocol,” *Proc. 1988 Int’l Conf. Parallel Processing*, vol. I, pp. 146-155, Aug. 1988.

[18] J. Archibald and J.-L. Baer, “Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model,” *ACM Trans. Computer Systems*, vol. 4, no. 4, pp. 273-298, Nov. 1986.

[19] J. Archibald, “A Cache Coherence Approach for Large Multiprocessor Systems,” *Proc. Int’l Conf. Supercomputing*, pp. 337-345, St. Malo, France, July 1988.

[20] A.L. Cox and R.J. Fowler, “Adaptive Cache Coherency for Detecting Migratory Shared Data,” *Proc. 20th Ann. Int’l Symp. Computer Architecture*, pp. 98-108, San Diego, Calif., May 1993.

[21] P. Stenstrom, M. Brorsson, and L. Sandberg, “An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing,” *Proc. 20th Ann. Int’l Symp. Computer Architecture*, pp. 109-118, San Diego, Calif., May 1993.

[22] W.-D. Weber and A. Gupta, “Analysis of Cache Invalidation Patterns in Multiprocessors,” *Proc. Third Symp. Architectural Support for Programming Languages and Operating Systems*, pp. 243-256, Boston, Apr. 1989.

[23] M. Brorsson and P. Stenstrom, “Visualizing Sharing Behavior in Relation to Shared Memory Management,” *Proc. 1992 Int’l Conf. Parallel and Distributed Systems*, pp. 528-536, Hsinchu, Taiwan, Dec. 1992.

[24] M. Brorsson and P. Stenstrom, “Modeling Accesses to Migratory and Producer-Consumer Characterized Data in a Shared-Memory Multiprocessor,” *Proc. Sixth IEEE Symp. Parallel and Distributed Processing*, pp. 612-619, Dallas, Tex., Oct. 1994.

[25] M. Brorsson, “SM-prof: A Tool to Visualize and Find Cache Coherence Performance Bottlenecks in Multiprocessor Programs,” *Proc. 1995 ACM SIGMETRICS and Performance ’95, Int’l Conf. Measurement & Modeling of Computer Systems*, pp. 178-187, Ottawa, Canada, May 1995.

[26] S.V. Adve, V.S. Adve, M.D. Hill, and M.K. Vernon, “Comparison of Hardware and Software Cache Coherence Scheme,” *Proc. 18th Ann. Int’l Symp. Computer Architecture*, pp. 298-308, Toronto, Canada, May 1991.

[27] S. Srbljic, “Model of Distributed Processing in Flexible Manufacturing Systems,” PhD dissertation, Inst. for Electronics, Faculty of Electrical Eng., Univ. of Zagreb, Croatia, Nov. 1990. (Work published in Croatian, original title: “Model distribuirane obrade u prilagodljivim proizvodnim sustavima”)

[28] S. Srbljic and L. Budin, “Analytical Performance Evaluation of Data Replication Based Shared Memory Model,” *Proc. Second IEEE Int’l Symp. High Performance Distributed Computing*, pp. 326-335, Spokane, Wash., July 1993.

[29] S. Srbljic, Z.G. Vranesic, and L. Budin, “Performance Prediction for Different Consistency Schemes in Distributed Shared Memory Systems,” *Proc. Third IEEE Int’l Symp. High Performance Distributed Computing*, pp. 295-302, San Francisco, Aug. 1994.

[30] V. Balasundaram, “A Mechanism for Keeping Useful Internal Information in Parallel Programing Tools: The Data Access Descriptor,” *J. Parallel and Distributed Computing*, vol. 9, pp. 154-169, June 1990.

[31] M.W. Hall, S.P. Amarasinghe. B.R. Murphy, S.W. Liao, and M.S. Lam, “Detecting Coarse-Grain Parallelism in Using an Interprocedural Parallelizing Compiler,” *Proc. Supercomputing ‘95*, 1995.

[32] S. Srbljic et al., “Models for Performance Prediction of Cache Coherence Protocols,” Technical Report CSRI-332, Computer Systems Research Inst., Univ. of Toronto, Canada, 1995. (ftp://ftp.cs.toronto.edu/pub/reports/ csri/332/332.ps.Z)

[33] J. Heinrich, *MIPS R4000 User’s Manual*. Prentice Hall, 1993.

[34] J.E. Veenstra, “Mint Tutorial and User Manual,” Technical Report 452, Computer Science Dept., Univ. of Rochester, May 1993.

[35] B. Boothe and A. Ranade, “Performance on a Bandwidth Constrained Network: How Much Bandwidth Do We Need?” *Proc. Supercomputing ‘93*, Portland, Ore., Nov. 1993.

[36] F. Dahlgren, “Boosting the Performance of Hybrid Snooping Cache Protocols,” *Proc. 22nd Ann. Int’l Symp. Computer Architecture*, pp. 60-69, Santa Margherita Ligure, Italy, June 1995.

[37] H.V. Leong and D. Agrawal, “Type-Specific Coherence Protocols for Distributed Shared Memory,” *Proc. 12th Int’l Conf. Distributed Computing Systems*, pp. 434-441, Yokohama, Japan, June 1992.

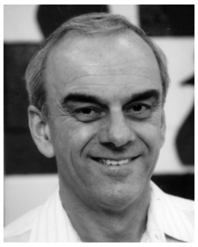
[38] S. Dwarkadas et al., “Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology,” *Proc. 20th Ann. Int’l Symp. Computer Architecture*, pp. 144-155, San Diego, Calif., May 1993.

[39] F. Dahlgren, M. Dubois, and P. Stenstrom, “Combined Performance Gains of Simple Cache Protocol Extensions,” *Proc. 21st Ann. Int’l Symp. Computer Architecture*, pp. 187-197, Chicago, Apr. 1994.

[40] J.E. Veenstra and R.J. Fowler, “The Prospects for On-Line Hybrid Coherency Protocols on Bus-Based Multiprocessors,” Technical Report 490, Computer Science Dept., Univ. of Rochester, 1994.

Sinisa Srbljic received his BS degree in electrical engineering in 1981, and MS and PhD degrees in computer engineering in 1985 and 1990, respectively, all from the University of Zagreb, Croatia. He is an associate professor at the University of Zagreb, School of Electrical Engineering and Computing. He was visiting the University of Toronto, Canada, from 1993 to 1995, where he worked on the NUMAchine multiprocessor project. As a visiting scientist from 1995 to 1996, he was working with the Advanced Technology Group of AT&T, USA, on caching of Internet objects in large distributed multimedia systems.

His research interests include parallel and distributed computer systems, compiler design, and performance evaluation.



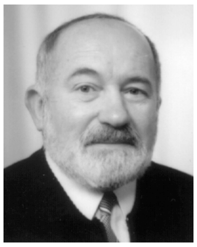
Zvonko G. Vranesic received the BASc, MASc, and PhD degrees in electrical engineering from the University of Toronto, Canada, in 1963, 1966, and 1968, respectively.

From 1963 to 1965, he worked as a design engineer for Northern Electric Co. Ltd., Bramalea, Ontario, Canada. In 1968, he joined the faculty of the Departments of Electrical Engineering and Computer Science at the University of Toronto, where he is now a professor. During the academic years 1977/78 and 1984/85, he was a senior visitor at the Computer Laboratory at the University of Cambridge, England, and at the Institut de Programmation at the University of Paris 6, France.

His research interests include computer architecture, VLSI systems, local area networks and many-valued switching systems. He has coauthored three books and has published more than 100 scientific papers. He was the chairman of the Third International Symposium on Multiple-Valued Logic in 1973 and of the 18th International Symposium on Computer Architecture in 1991.

Michael Stumm received a diploma in mathematics and a PhD in computer science from the University of Zurich in 1980 and 1984, respectively. He is a professor in the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Toronto, Toronto, Canada.

Dr. Stumm's research interests are in the areas of computer systems, in particular, operating systems for distributed and parallel systems. He is a member of the IEEE Computer Society and the ACM.



Leo Budin received the BE, MS, and doctoral degrees, all in electrical engineering, from the University of Zagreb, Croatia. In 1962, he joined the research and teaching staff of the Faculty of Electrical Engineering of the University of Zagreb. He spent the academic year 1968-1969 as an Alexander von Humboldt scholar, at the University of Erlangen-Nuernberg, Germany, and the academic year 1979-1980 as a Fullbright scholar at the University of Illinois, Urbana-Champaign. In 1982, Dr. Budin was appointed a professor of electrical and computer engineering and computer science at the University of Zagreb. He is coauthor of the textbook *Computer-Aided Analysis* (in Croatian). He served for several years as the editor-in-chief of the *Journal of Computing and Information Technology* (*CIT*). His current research interests include distributed systems, real-time systems, soft computing, and problem solving environment in engineering. He is a member of the IEEE and the ACM.