Analytic Evaluation of Shared-Memory Architectures for Heterogeneous Applications

Daniel Sorin
Jonathan L. Lemon
Derek L. Eager
Mary Vernon

Technical Report #1404 b

January 2000

Analytic Evaluation of Shared-Memory Architectures for Heterogeneous Applications

Daniel J. Sorin[†], Jonathan L. Lemon[†], Derek L. Eager[‡], and Mary K. Vernon[†]

[†]Computer Sciences Department
University of Wisconsin - Madison
{sorin,lemon,vernon}@cs.wisc.edu

[‡]Department of Computer Science
University of Saskatchewan
eager@cs.usask.ca

Technical Report #1404b, Computer Sciences Dept., Univ. of Wisconsin - Madison

Abstract

This paper develops and validates a new analytical model for evaluating the performance of shared memory systems with ILP processors. First, we instrument SimOS to measure the per-processor parameters for such a model, and we find a perhaps surprisingly high degree of processor memory request heterogeneity observed in the SimOS workload measures. Second, we create a model that captures such heterogeneous processor behavior, which is important for analyzing memory system design tradeoffs. Highly bursty memory request traffic and lock contention are modeled in a significantly more robust way than in previous work. With these additions, the model is applicable to a wider range of architectures and applications. Although the features increase the model complexity, it is a useful design tool because the size of the model input parameter set remains manageable and the model is still several orders of magnitude quicker to solve than detailed simulation.

Validation results show that the model is highly accurate, producing heterogeneous per-processor throughputs that are usually within 5% and always within 13% of those measured by detailed simulation with SimOS. We show several applications of the model to studying architectural design issues and the interactions between the architecture and the heterogeneous applications.

1 Introduction

Computer architects traditionally use detailed simulation to evaluate architecture performance trade-offs. Detailed simulation of parallel architectures with complex modern processors usually entails a cycle-by-cycle simulation of each processor that precisely captures significant behavior such as out-of-order instruction issue and speculative instruction execution that can greatly affect system performance. Detailed simulation is time-consuming, though. For example, detailed simulation of an 8-processor shared memory architecture, running a single parallel FFT code with a small input dataset, can take hours on a Sun UltraSPARC system, even though only seconds of the system execution time are actually simulated.

To design a given memory system architecture, one would like to evaluate alternative memory system architectures for dozens if not hundreds of commercial applications and workloads that might be expected to run on the system. Thus, more efficient evaluation methods that can aid in culling the system design space are highly desirable. Analytical models offer the possibility of efficiently computing performance estimates that can be useful in identifying the most promising regions of the architectural design space, which can then be explored more fully using the detailed simulation approach. The key issue is devising an analytical model that is sufficiently accurate for this purpose, over the range of workloads of interest. If such a model can be constructed, it also offers the opportunity to explore how the memory system architecture performs for hypothetical changes in the memory request behavior of the executing workload. Experienced system architects may be interested in exploring these issue, but this is difficult to do with detailed simulation of specific benchmarks.

Three recent papers have developed analytical models that contain some of the significant features of complex modern shared memory multiprocessor architectures [1, 11, 10]. Of these models, the previous ("SM-ILP") model by Sorin et al. [10] is the only model that (1) includes the impact of instruction window size and dependences between memory accesses, which cause a processor to block after a dynamically changing number of memory requests, and (2) has been validated against detailed simulations of applications running on a parallel shared memory architecture. The model has a number of significant features. First, it is based on a relatively small set of input parameters that are sensitive to changes in the processor and associated cache architecture, but are relatively *insensitive to changes in the rest of the memory system architecture*. Second, the model captures the key characteristics of a complex modern processor architecture which are important for memory system design, such as speculative memory requests and complex processor blocking behavior. Third, it produces results for variations in the memory system architecture in a few seconds, and these results have been shown to predict processor throughput, measured in instructions per cycle, that is within 1-12% of the detailed simulation estimates for several Splash-2 applications [12] running on the RSIM architecture [7].

The previous SM-ILP model also has at least three significant deficiencies. First, it assumes that each processor is statistically identical with respect to memory request behavior, and that, for each processor, its remote memory requests are equally likely to visit each of the remote memory modules. These homogeneity and uniformity assumptions, which were sufficiently accurate for several SPMD applications running on RSIM, preclude the model from being used for many application/platform combinations that might be of interest. For example, many irregular applications can be expected to have heterogeneous processor behavior. Furthermore, an application's dataset and/or the work may not be distributed evenly among the nodes. Heterogeneous memory request behavior can have a disproportionate impact on system throughput, due to non-linear queueing effects in the memory system. Thus, it is important for a model that supports memory system design to capture such behavior. Second, the model overestimates the fraction of time that a processor is productively executing, and it underestimates the average waiting time at the memory system resource first visited by a request that cannot be satisfied by the processor cache hierarchy. These errors cancel each other in the estimated system throughput measure, and thus the overall system throughput estimates were found to be highly accurate in the validation experiments [10]. However, more accurate memory system waiting time estimates are needed if the model is to be applied reliably for identifying memory system bottlenecks. Finally, although the model uses a basic set of input parameters for computing processor throughput, it uses measured average lock waiting times to compute total application running time from the estimated processing rate. Since lock contention delays are affected by delays in the memory system, this input parameter is dependent on the output values of the model. A more basic model of lock contention is needed to compute the application running time.

In this paper, we derive and validate a new model that captures all of the above behavior for shared memory system architectures with complex modern processors. Furthermore, we develop this model for the architecture and workload that is simulated in SimOS [8]. This tests the robustness of the basic analytic approach for a significant change in the memory system architecture, including a different memory consistency model, and for workloads that include the (SGI IRIX) operating system as well as the Splash applications. One of the results of these new validations is that the SimOS workloads have quite different behavior than RSIM workloads with respect to the memory system.

Specific contributions of this paper include:

- Measured parameters are provided that illustrate the types of heterogeneity that occur in the workloads that are simulated using SimOS. These parameter values show that, although the SGI IRIX operating system executes uniformly across all of the processors, Splash-2 applications that are generally considered to be homogeneous have highly heterogeneous memory request behavior. More detailed measures reveal several causes of the heterogeneity.

- The new model provides parameters for specifying the heterogeneous memory request behavior. That is, each processor can have different mean time between memory requests, distribution of the number of outstanding requests when the processor blocks, and so forth. Moreover, the memory requests from each processor can have a different distribution of destinations for requests to remote nodes. The way that processor blocking behavior is modeled in [10] is incompatible with modeling heterogeneous processor behavior, and thus this important aspect of the previous model must be modified in creating the new model. A key question addressed in this work is whether the model can remain tractable, both from a programming effort standpoint and a solution-time/convergence standpoint, when the memory request heterogeneity is represented. Another key question is how well the heterogeneous model will validate with respect to individual processor throughput estimates and with respect to estimated mean queueing delays in the memory system.

- Based on new AMVA methods proposed in recent work [3], we develop equations that more accurately model the processor utilization and the mean waiting time at memory system resources when the time between memory requests that miss in the second level cache is highly variable. The key issues here are how to adapt the new AMVA methods to the context of the memory system architecture and Splash-2 applications, and whether the methods will prove to be accurate in this more complex model.
- We develop a submodel that accurately estimates mean lock access delays from fundamental input parameters that are independent of changes in the memory system architecture (below the processor cache hierarchy).

While modeling the additional behaviors increases the model's size and complexity, the model solution time is still on the order of a couple of seconds. The number of input parameters is increased, but not to the point of being unwieldy.

Validations in this paper show that the new model predicts heterogeneous processor performance, from a manageable set of input parameters, that agrees with detailed SimOS estimates for a set of benchmark applications running on the SimOS architecture. The percentage difference between the performance computed by the model and the performance reported by SimOS for each processor is typically within 5% and always less than 13% over the validation experiments performed in this work. The validation results and example model applications also show that modeling heterogeneity is important for achieving high model accuracy. Thus, this capability is essential both to achieving wider applicability of the model and for increasing confidence in using the model to find the promising regions of the memory system architecture design space that should be investigated using detailed simulation.

Three examples are provided to illustrate the use of the new model. One example illustrates the use of the model to evaluate alternative memory system designs under a heterogeneous workload. The two other examples provide estimates of the (maximum) performance gain that can be achieved if the application were "tuned" to remove the heterogeneity that is observed in the measured parameters for the application.

Like the previous analytic models, the new model input parameters are derived from a detailed simulation of an application or workload running on a given parallel processor architecture. However, the model input parameters are carefully chosen so as to be insensitive to large changes in the memory system latency below the processor cache hierarchy. Thus, as shown in [10], the analytic model can accurately predict system performance when various memory system components below the processor cache hierarchy are modified. Consequently, the analytic model can be used to quickly cull the design space for this part of the memory system, for both measured workload parameters and hypothetical modifications to measured workload parameters, thereby greatly reducing the size of the design space that needs to be explored using simulation.

The rest of this paper is organized as follows. Section 2 describes the system that we are modeling and reviews the previous system model. Section 3 discusses the model extensions for heterogeneous applications and provides measured application parameters that illustrate the types of heterogeneity that occur in the SimOS benchmarks. Section 4 develops new modeling approaches for bursty traffic and synchronization in shared memory multiprocessors. Section 5 presents the model validations and discusses applications of the model. Finally, Section 6 summarizes the paper and discusses future research.

2 Background

2.1 System Architecture

The system of interest in this paper is the system modeled by SimOS [8], a detailed simulator developed at Stanford against which the new model will be validated. The architecture, which is similar to that of the Stanford FLASH [5] and the SGI Origin [6], is a cache-coherent, sequentially consistent shared-memory multiprocessor

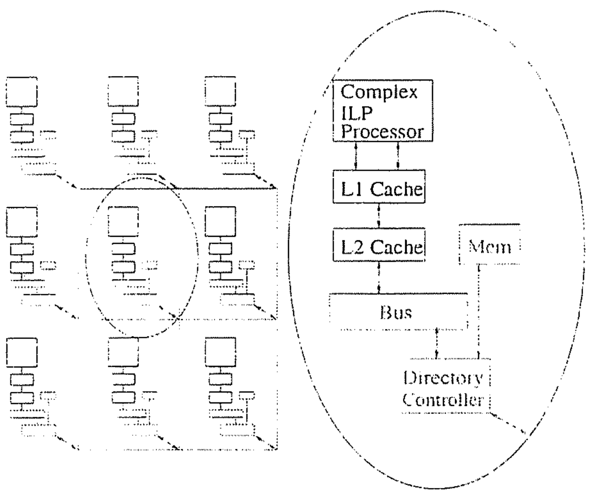

Figure 1. System Architecture

system, as shown in Figure 1. This architecture differs in several key ways from the architecture modeled by RSIM [7] and the SM-ILP model, as discussed below.

The MIPS R10000 processor, modeled by SimOS's MXS simulator, exploits instruction level parallelism using multiple functional units, out-of-order execution, non-blocking loads, and speculative execution. Instructions are fetched into the instruction window, and they are issued to the functional units after all of their input data dependences are satisfied. Speculative execution is used for (temporarily) unresolved control dependences for up to four branch instructions. The instructions are fetched into and retired from the window in program order, but they may be issued to the functional units out of program order.

An instruction can retire from the instruction window only after it completes execution. A key implication of this requirement is that when a load reaches the top of the instruction window, retirement must stall if the data has not yet returned. SimOS differs from RSIM in that, since SimOS is a sequentially consistent architecture, stores in SimOS only issue to the memory system when they reach the top of the instruction window.

The caches use miss status holding registers (MSHRs) to track the status of all outstanding misses [4]. Misses to the same cache line are *coalesced* in the MSHRs; only one memory request is generated for such coalesced misses.

All traffic to or from a remote node goes through the directory controller (DC). Traffic into the node that only requires accessing the directory does not require use of the bus. SimOS models the memory bus and the interconnection network using fixed latencies that account for service time as well as estimated contention delay. SimOS also differs from RSIM in that it does not model a separate network interface since the DC serves that purpose.

Cache coherence is maintained by a fairly standard three-state (MSI) directory-based invalidation protocol. Unlike the RSIM architecture, there are no cache-to-cache transfers; instead the home node is responsible for collecting invalidations before acknowledging a request for exclusive permission.

Table 1 defines the system architecture parameters, including the values that are used in the validation experiments in Section 5.1. Latencies are in units of CPU cycles for the 200 MHz R10000 processor. Note that memory access is overlapped with directory access; thus, there is just one parameter for that access latency, $S_{DC_{long}}$.

2.2 Previous Model

The previous model developed by Sorin et. al [10] is a customized Approximate Mean Value Analysis (AMVA) model of homogeneous applications running on the release consistent shared-memory multiprocessor architecture modeled by the RSIM simulator, which is also based on an R10000-like processor. As claimed in that paper, it is not very difficult to modify that model for other shared memory multiprocessor architectures. Below we provide a brief overview of the model and the modifications for the SimOS architecture. Appendix A outlines the equations used in the homogeneous model. For more detail about the model equations, the reader is referred to [9].

parameter	description	value
N	number of nodes	
m	memory modules per node	1
M_{hw}	number of MSHRs	8
S_{bus}	bus latency	15
S_{DC}	DC latency	5
$S_{DC_{long}}$	long DC latency	20
S_{net}	network latency	30

Table 1. System Architecture Parameters

The homogeneous model input parameters are summarized in Table 2. These parameters characterize the memory request behavior (between any two barriers) of an application running on the architecture. The first three parameters characterize the rate of requests to the memory system, the burstiness in the memory interrequest times, and the number of outstanding requests when the processor blocks due to a memory request that cannot be retired. The rest of the parameters characterize the types of requests that are being issued to the memory system. Sorin et al. observe that these input parameters are sensitive to instruction window size, processor architecture, organization and size of the processor cache hierarchy, and various aspects of the application code and compiler, but are relatively insensitive to memory system latencies below processor cache hierarchy [10].

The processor and cache subsystem are modeled as a black box that - when not completely stalled - issues memory requests at a given rate and with a given coefficient of variation in interrequest times. The model computes the overall mean system residence time for a memory request, including mean delays and service times at the DCs, split-transaction memory buses, and in the interconnection network, as reviewed in Appendix A. Two submodels that each use the same set of equations for overall mean system residence times are used to compute two types of processor stall time, respectively. One submodel computes stall time due to reads that cannot be retired until data returns from memory. The other submodel computes stall time due to MSHRs that are fully occupied by write requests. The model also accounts for reads that coalesce in the MSHRs with writes to the same line. Iteration between the two submodels is necessary because the each stall time estimate is needed to compute the other stall time estimate.

We changed the previous model in two ways due to differences between the modeled architectures. First, the routing of requests (e.g., the path that a read request takes through the memory system) is different for SimOS since the memory system resources and protocol are different. The second change is due to the different memory consistency model. To model the sequentially consistent SimOS system where write requests are synchronous, we observe that both types of stall times can be computed in a single model that has the number of MSHRs as the number of memory requests that can be issued before the processor stalls. Thus, iteration between two submodels is not required for the SimOS architecture.

2.3 Methodology

As mentioned above, previous work has verified that the homogeneous model input parameters are, to first order, insensitive to changes in memory system latency below the processor cache hierarchy. Thus, for the new model developed in this paper, we start with the same input parameters for each processor, but allow each processor to have a different value for each parameter. One question is whether these parameters are sufficient for accurately computing processor throughputs and mean delays in the memory system for heterogeneous workloads. This question will be investigated by comparing the estimates against the performance measures that are given by SimOS. If the analytic estimates are accurate for a range of heterogeneous workload parameter sets, we can then expect to be able to vary memory system architecture parameters without significantly altering the accuracy of the analytic model.

The detailed simulator needs to be run once to obtain the set of parameters for a given application/workload of interest executing on a given processor and cache architecture of interest. However, the parameters can then be used with the analytic model to explore the impact of various architecture parameters, including the number of MSHRs, the interconnect design, and main memory organization. The detailed simulator may also be used to

Parameter	Description
τ	Average time between read, write, or upgrade requests to memory, not counting the time when the processor is completely stalled or is spin-waiting on a synchronization event
CV_τ	Coefficient of Variation of τ
f_M	Fraction of processor stalls that occur with $M = 1, 2, ...$ outstanding requests in the MSHRs
$P_{read}, P_{write}, P_{upgrade}$	Probability that a memory request is a read, write, or upgrade
P_{wb}	Probability that a read or write request causes a writeback of a cache block
$P_{L\|x}$	Probability directory is local for a type x transaction; x=read, write, upgrade, write-back
$P_{M\|x,y}$	Probability home memory can supply the data for a type x, y request; x=read, write; y=local home, remote home
$P_{4hop\|x\&not-memory}$	Probability that a request of type x to a remote home is forwarded to a cache at a third node; x=read,write
X	Average number of invalidates caused by a write or upgrade to a clean line

Table 2. Model Application Parameters

verify results for specific memory system architectures below the processor cache hierarchy that are identified as having the highest performance by the analytical model. However, by varying the memory system architecture parameters and using the input parameters measured from detailed simulation, the analytical model can be used to quickly explore the memory system design space for a given processor/cache architecture and workload. Moreover, the designer can vary workload input parameters (e.g., probability that a transactions takes 3 hops) to test the system against specific synthetic workloads, which is difficult to do by altering the actual workload.

3 Heterogeneity

The previous model assumes that all processors have statistically similar behavior with respect to the memory system and that each processor's local/remote memory accesses are uniformly distributed across the local/remote memory modules. In the homogeneous model, each processor has the same input parameters, as shown in Table 2, and no input parameters are needed for frequencies of access to each memory module.

In Section 3.1, we examine the per-processor model input parameters obtained for several Splash-2 benchmarks using SimOS, to see what types of heterogeneity occur in the processor memory requests for those applications. In Section 3.2, we discuss how the analytic SimOS architecture model computes performance estimates for this heterogeneous processor behavior.

3.1 SimOS Application Parameters

Figures 2, 3, and 4 illustrate the processor heterogeneity in several key parameters for particular barriers (i.e., inter-barrier phases) of a few SPLASH-2 [12] benchmarks, each running on eight processors, as measured by SimOS. For each input parameter shown, the eight bars represent the values of that parameter for each of the eight processors, divided by the value of that parameter when measured over all eight processors. The heterogeneity in various measures for particular barriers is also summarized in Table 3. The parameter M_{ave} refers to the average of the distribution f_M.

The degree of parameter heterogeneity, such as in the measures of τ and $P(L|w)$ in the figures and Table 3, is perhaps higher than might be expected. Some (irregular) applications, such as Radiosity, are inherently heterogeneous, and thus the per-processor memory request measures simply quantify the degree of memory request heterogeneity that occurs in those applications. However, as can be seen from the figures and the table, significant memory request heterogeneity is also present for SPMD applications such as FFT and Radix, that are generally thought to be homogeneous and were observed to have homogeneous memory request behavior in RSIM [7].

In an attempt to determine the source of the processor memory request heterogeneity for these SPMD applications on SimOS, we separated the statistics into application and kernel statistics for each processor. For example,

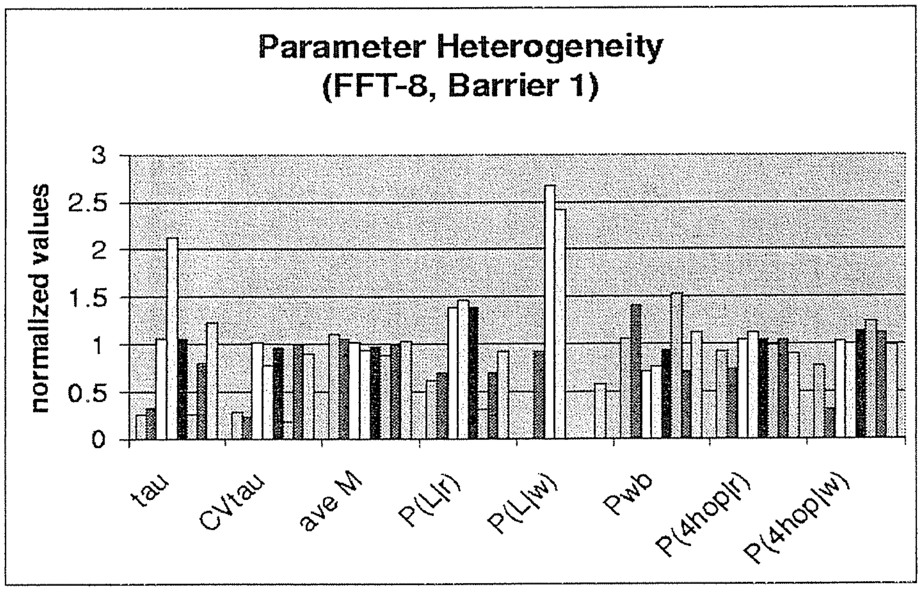

Figure 2.

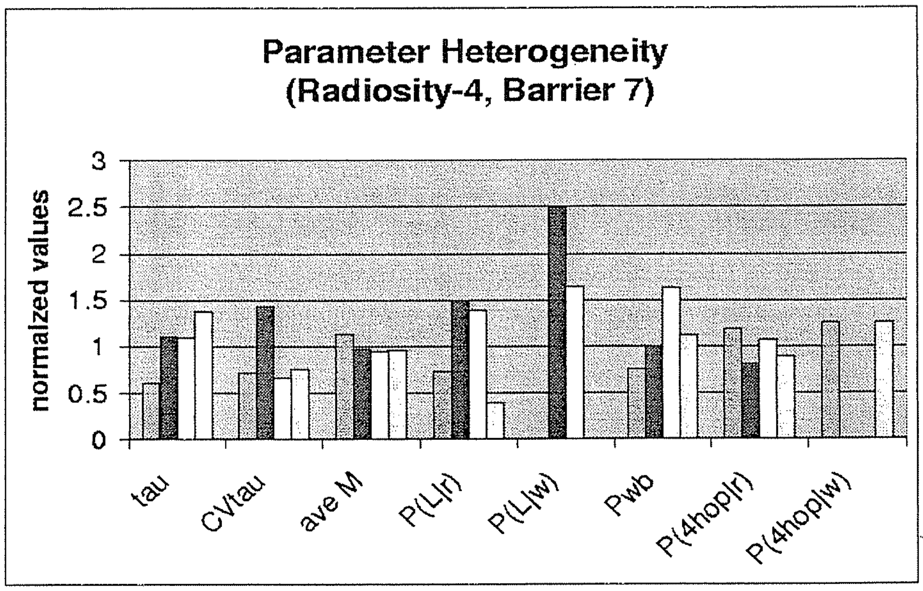

Figure 3.

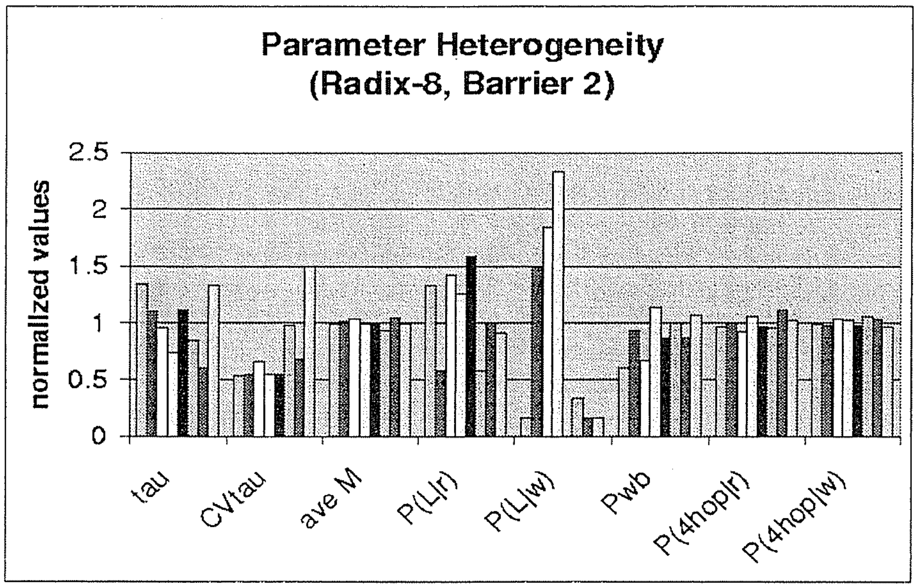

Figure 4.

benchmark	barrier	parameter	mean	min	max
FFT-8	1	M_{ave}	1.39	1.23	1.53
	2	τ	60	42	78
	3	$P(L\|u)$	.13	.01	.30
Radiosity-4	3	τ	78	48	122
	7	$P(4hop\|r)$	.66	.53	.78
Radix-8	1	$P(L\|w)$	.12	.01	.33
	3	τ	103	31	210
	4	$P(L\|r)$	.13	.04	.25

Table 3. Parameter Heterogeneity

the number of memory requests per processor for the transpose phase (barrier 2) of FFT-4 is shown in Table 4. As expected, the number of memory requests that are issued when a processor is executing the application are quite homogeneous across the processors, with a coefficient of variation of 0.02. What is perhaps surprising is both the high numbers of memory requests that are issued in kernel mode (larger, in fact, than the numbers of application requests) as well as the heterogeneity in this number across processors, as shown by a coefficient of variation of 0.27. For the SPMD applications, other parameters like CV_{τ}, exhibit similar heterogeneity in the kernel while remaining homogeneous in the application. Note that in Table 4, the number of memory requests appears to be correlated with the processor number, but this correlation is just coincidental and did not occur with any higher than random frequency in the results that we obtained for different barriers in the FFT application and for different applications.

A key point is that the kernel executes uniformly across the processors in the SimOS architecture, and thus the kernel memory request statistics are fairly homogeneous across the processors if measured over a long time interval, such as the execution time for the entire application. However, the memory request heterogeneity observed in the measured intervals between each barrier will impact memory system performance, and, therefore, the heterogeneity in this timeframe must be measured and represented in the inputs to the model.

Beyond the heterogeneity inherent in some applications or caused by kernel behavior, some applications can

	cpu 0	cpu 1	cpu 2	cpu 3
overall	461	613	781	787
application	216	219	231	219
kernel	245	394	550	568

Table 4. Memory Requests for FFT-4 Barrier 2

exhibit heterogeneous behavior if they have not been "tuned" to run on a particular architecture and runtime system, which occurs frequently in practice. Heterogeneity in the measured model input parameters can point to the need for such tuning, and even indicate what types of tuning are needed. For example, in barrier 1 of FFT-8, three processors have small relative values of τ when the application is executing, indicating that they have especially high level 2 cache miss rates. Those same processors (and one additional processor) have a relatively low probability of local memory access for read and/or write requests. Thus, examining the data layout or comparing the code that runs on those three processors against the code that runs on the other five processors, looking for differences that might cause these effects, may lead to some insight about how to improve performance. Similarly, the heterogeneity in the probability that a write request is local for Radix suggests that data layout should be examined for that application as well.

In general, although there may be intuition that particular applications will exhibit heterogeneous behavior of some form, intuition alone is generally insufficient to estimate the magnitude of the heterogeneity in particular statistics of interest, its magnitude relative to the heterogeneity induced by kernel activity, or the extent to which performance might be improved by particular types of application tuning. Measurements of heterogeneity, and the use of models that capture its performance impacts, can provide answers to such questions.

The figures and the table indicate that heterogeneity occurs in practice for every model input parameter. Two parameters, though, have notably less extreme heterogeneity: (1) the average of the f_M distribution (i.e., the average number of memory requests that are outstanding when the processor blocks because a load or store cannot be retired) and (2) the probability that a remote read request for a dirty block requires invalidating or downgrading the line in a cache at another remote node ($P_{4hop|r}$). However, system performance can be sensitive to the values of these parameters. Thus, the model extensions for processor heterogeneity will allow each processor to have its own value for each of the input parameters.

3.2 Modeling Heterogeneity

Given that (1) processor heterogeneity (and memory access non-uniformity) occur quite frequently in practice, and (2) we can expect the heterogeneity to have a non-linear impact on queueing and memory system performance, it is important to develop models that enable computation of system performance measures for heterogeneous workloads. We will show in Section 5.1 that modeling heterogeneity is critical for achieving accurate results.

Three features are needed in order to model heterogeneous processor behavior. First, new model inputs are required. Specifically, each of the model input parameters in Table 2 is measured for each processor. Also, modeling memory access non-uniformity requires additional parameters that specify, for each processor and type of remote memory request, the probability that the request will be directed to each other node.

Second, we require a new model of the varying number of outstanding requests, M, that are issued by the processor before the processor blocks waiting for a memory system response. The SM-ILP model computes system throughput for each possible value of M with the same value of M at every processor, and then computes a weighted sum of these throughputs, where the weights are computed from the measured distribution of M, f_M. This weighted sum approach may be valid if the processors generally have the same value of M at the same time, as might be true for the homogeneous SPMD applications that were modeled in [10]. However, the weighted sum technique is difficult to apply in the case where different nodes have different distributions of M. Computing the throughput for every possible combination of M values at the different processors, as well as computing the weighting factor for each such throughput, is prohibitively complex. An alternate approach is to use the average value of M at each node. The drawback of using the average value is that it may fail to capture the non-linear effects of varying M; thus it may not capture a broad distribution of M accurately.

In Table 5, we compare the accuracy of the estimated processor throughput for the weighted sum approach

										Processor Throughput (IPC)		
app	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	M_{ave}	weighted	ave M	actual
erle16	.65	.17	.09	.08	0	0	0	0	1.64	1.35	1.40	1.45
fft-opt16	.42	.17	0	0	.34	0	0	0	2.72	1.53	1.61	1.58
fft16	.53	.47	0	0	0	0	0	0	1.47	1.46	1.46	1.39
lu-opt8	.12	.06	.06	.06	.06	.06	.07	.49	5.83	2.46	2.19	2.41
lu8	.51	.49	0	0	0	0	0	0	1.50	1.96	1.91	1.91
radix8	.99	.01	0	0	0	0	0	0	1.01	1.76	1.74	1.64
water16	.73	.25	.01	0	0	0	0	0	1.33	1.87	1.62	1.74

Table 5. Accuracy of Processor Throughput (IPC) Estimates for Weighted Sum vs Average M

against simply solving the model once for the average value of M for some of the homogeneous applications that were simulated using RSIM. These results provide some evidence that, for current window sizes and applications, the average M approach achieves similar accuracy to the weighted sum approach. We compute average M from the measured f_M, where the f_M are measured assuming an infinite number of MSHRs, and we allow each processor to have its own average value of M which is limited by the number of MSHRs, M_{hw}.

The third change is to modify the model equations [9, 13] to compute performance metrics at each resource within the memory system for heterogeneous processor loads. With homogeneous behavior, it is only necessary to compute performance metrics for a single generic resource of type i and a single generic processor load: the total utilization of the particular memory resource, for example, can then be obtained simply by multiplying by the number of processors. This leads to model complexity on the order of the number of types of resources, as can be seen in the equations in Appendix A. For heterogeneous processor loads, in contrast, each processor may have differing memory referencing behavior and thus may contribute to differing extents to utilization and contention at each resource. Computing the N^2 interactions of each processor on each memory system resource increases the complexity of the model equations by a factor of N^2, which leads to a key question about whether the iterative model will converge in practice. This issue is addressed in Section 5.1. Efficient coding methods (for both the homogeneous and heterogeneous model) limit the size of the model (measured in C++ code), though, to only about twice that of the homogeneous model. The accuracy improvements that may be obtained by accurately modeling heterogeneous memory request behavior when it exists, rather than assuming homogeneity, are also illustrated in Section 5.1.

All of the details of the heterogeneous AMVA equations can be found in [13]. At a high level, the heterogeneous equations have form similar to the homogeneous equations. Terms require extra indices (given in brackets) to indicate the node of the customer and/or the destination. Thus, a utilization term such as $U_{dc_{loc}}$ (the mean utilization of the local DC) becomes $U_{dc_{loc}}[i]$ (the mean utilization of the DC of node i by local customers) to reflect the fact that the utilization of the local DC is different for different nodes. The probabilities of the transaction types (e.g., local read or 3-hop write) use an index in a similar fashion. For example, we now have that the probability of transaction y *at node* i is $P_y[i]$. So, the total mean residence time for a customer from node i, $R[i]$, is equal to the sum of its mean residence times at the processor (pe), buses, network, and directory controllers (dc).

$$R[i] = R_{pe}[i] + R_{bus}[i] + R_{net}[i] + R_{dc}[i]$$

Examining the DC portion of this equation highlights the differences in the equations between the heterogeneous and homogeneous models. Mean DC residence time is equal to the mean residence time at the local DC plus the mean residence time at the DCs of the other nodes.

$$R_{dc}[i] = R_{dc_{loc}}[i] + \sum_{\substack{j \\ j \neq i}} R_{dc_{rem}}[i][j]$$

Only focusing on the mean residence time at the remote DC, it is the sum of the mean residence times over the different types of transactions, denoted by a subscript of y.

$$R_{dc_{rem}}[i][j] = \sum_{y} R_{dc_{rem,y}}[i][j]$$

The mean residence times of individual transaction types are equal to the probability of the transaction type ($P_y[i]$) times the visit count ($V_{dc_{rem_y}}[i][j]$) times the sum of the mean waiting time ($W_{dc_{rem}}[i][j]$) and the service time at the DC (S_{dc}):

$$R_{dc_{rem,y}}[i][j] = P_y[i]V_{dc_{rem_y}}[i][j](W_{dc_{rem}}[i][j] + S_{dc})$$

All of the terms in the above equation are inputs except for the waiting times. The equation for mean waiting time at a remote directory is as follows:

$$W_{dc_{rem}}[i][j] = W^{others}_{dc_{rem}}[i][j] + W^{rem}_{dc_{rem}}[i][j]$$

$W^{others}_{dc_{rem}}[i][j]$ is the mean waiting time of a node i customer at the DC of node j due to traffic from all nodes other than node j. $W^{rem}_{dc_{rem}}[i][j]$ is the mean waiting time of a node i customer at the DC of node j due to traffic from node j. Only breaking down $W^{rem}_{dc_{rem}}[i][j]$ further, we have that

$$W^{rem}_{dc_{rem}}[i][j] = \sum_{y} (W^{rem.y}_{dc_{rem}}[i][j])$$

The following equations are for the mean waiting times of specific transaction types. Thus, $W^{rem.y}_{dc_{rem}}[i][j]$ is the mean waiting time by a node i customer at node j's DC due to node j traffic for transactions of type y. Mean waiting time due to a single node j customer equals $\frac{R_{dc_{loc,y}}[j]}{R[j]} - U_{dc_{loc,y}}[j]$ (the probability that a customer is in the queue but not in service) times the service time, plus $U_{dc_{loc,y}}[j]$ (the probability that a customer is in service) times the mean residual life of a customer in service. Therefore, to get the total mean waiting time, we multiply by the number of node j customers, $M[j]$.

$$W^{rem.y}_{dc_{rem}}[i][j] = M[j]\left[\left(\frac{R_{dc_{loc,y}}[j]}{R[j]} - U_{dc_{loc,y}}[j]\right) S_{dc} + U_{dc_{loc,y}}[j]\left(\frac{S_{dc}}{2}\right)\right]$$

Lastly, we have the equation for the mean utilization of node i's DC by a local customer.

$$U_{dc_{loc,y}}[i] = \frac{P_y[i]}{R[i]}(V_{dc_{loc,y}}[i]S_{dc})$$

Modeling the other resources in the system is similar to what has been shown here for the DC.

4 Modeling Bursty Memory Requests and Lock Contention

In this section, we present two key model extensions. A recent paper [3] proposes new AMVA techniques for modeling bursty arrivals in simple queueing networks. In Section 4.1, we show how these ideas can be applied to model bursty memory system traffic observed in the SimOS workload measures. In Section 4.2, we develop a method for computing lock synchronization from basic model input parameters. Bursty memory requests and lock synchronization play significant roles in system performance, and thus it is important to model them accurately.

4.1 Burstiness

In the previous SM-ILP model, the mean residence time of a "customer in service" at the processor (i.e., a memory request about to be generated) is computed using a specially derived interpolation for mean residual life. In this paper, we employ a more accurate AMVA equation (called "AMVA-decomp" in [3]) for mean residence time at the processor queue. In addition, we adapt the AMVA techniques in [3] to accurately compute the mean wait at the "downstream" queues (i.e., the local DCs in the SimOS architecture) which have bursty arrivals from the processors, and thus increased queueing activity.

The AMVA-decomp equation accurately computes mean residence time for a 2-stage hyperexponential server. That is, with probability p a given customer has a "small" mean service time, τ_a, and with probability $1-p$ the customer has a "large" mean service time, τ_b, where $\tau_a < \tau_b$ and $\tau = p\tau_a + (1-p)\tau_b$. The key question in applying this technique for the processors in the heterogeneous model is how to obtain the parameters of a suitable hyperexponential distribution. Two constraints on the distribution are the measured mean service time (τ) and the coefficient of variation in the service time (CV_τ). However, this is an underconstrained problem. To apply this technique to modeling heterogeneous bursty processors, we measure a third parameter for each processor, τ_a, which is set to be equal to the minimum sampled value of τ. Using τ, CV_τ, and τ_a, we solve for τ_b and p. The model is more sensitive to the value of τ_a than to that of τ_b, especially in the high variance cases where $\tau_a << \tau_b$. Thus, it is preferable to assign τ_a to a measured value and allow τ_b to float. Moreover, the true best value of the "small" mean is likely to be near the measured minimum value of τ, while there is no measured value that corresponds well to τ_b.

In the model of bursty requests at the downstream queues, there are bursts of arrivals and intervals between bursts in which there are no arrivals. This scenario is characterized by three parameters:
k, the average number of customer arrivals within a burst,
I_i, the mean interarrival time within a burst, and
I_o, the mean time between bursts.

In applying the bursty request model to the local DCs in the SimOS architecture, the key question again is how to map the requisite parameters to observable quantities in the system. There are two constraints in determining values for these three parameters. First, the coefficient of variation in the interarrival time is determined by the coefficient of variation in the service time at the processor. Second, the throughput at the downstream queue, and thus the mean interarrival time, is determined during the AMVA solution. We create a third constraint by setting I_i equal to the value of τ_a, since it is reasonable to assume that interarrival time during a burst would be similar to the value of a short service time at the processor. We also assume that downstream burstiness is only caused by requests from the local processor. The superposition of requests from other processors will, on average, be less bursty.

Solving the model with the burstiness equations initially led to some convergence problems. To ensure that the model converges requires some careful choices of initial values and bounds checks to make sure that values produced during iterative solutions are reasonable. For example, k cannot be allowed to be larger than the total number of local customers.

4.2 Lock Contention

The previous SM-ILP model measured average lock waiting times, which are affected by the memory system architecture, instead of computing these performance measures from more basic parameters that are independent of the memory system architecture. In this paper, lock synchronization effects are computed from basic inputs with a separate lock contention model. The context that we consider allows the application program to queue for a lock while occupying a processor. Symmetrically, while holding the lock, the program can queue for memory system resources or the processor. In addition, the program can release the lock while still holding the processor and it can complete service at a memory system resource or processor while still holding the lock. We assume for now that only one lock is held at a time, but we will describe later how to relax this assumption.

In the lock contention model, there are N customers, representing the processors that are vying for the locks. There are First-Come-First-Served (FCFS) queues for each of the locks that have non-negligible contention, and there is a delay center which represents the execution time while not accessing or holding any of the locks. An example lock queueing model is shown in Figure 5.

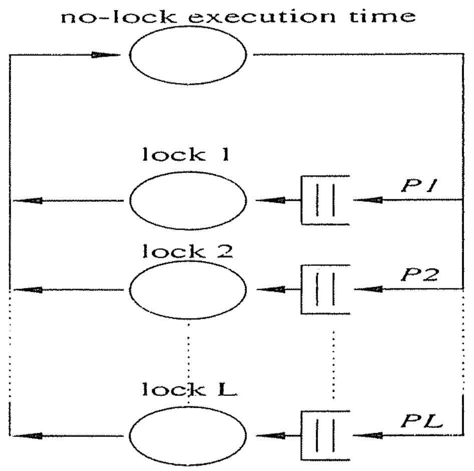

Figure 5. Example Lock Contention Model

The basic parameters used to characterize lock behavior in the example are the following:
L: number of locks that have non-negligible contention
$r_{nolock,j}$: average number of memory accesses made by processor j between lock requests
$P_{lock_i,j}$: probability that a lock request from processor j is for lock i
$r_{lock_i,j}$: average number of memory requests by processor j while holding lock i

To incorporate the effects of lock contention in the architecture model, we iterate between the architecture model and the lock model. The service times in the lock model are derived from the above basic parameters, and mean residence times computed from the architecture model. The service times at the processors in the architecture model are inflated so as to reflect lock waiting times computed from the lock model. This iterative solution technique again increases the complexity of the model, an issue that will be addressed in Section 5.1. It also causes solution time to increase slightly, but it is still on the order of seconds. Moreover, this iterative technique can be generalized for specific cases of nested lock requests by having a separate lock contention model for the locks at each level of the lock hierarchy and iteratively solving the lock models along with the architecture model. The details of the lock contention equations can be found in [13].

5 Model Validation and Applications

5.1 Model Validation

In this section, we present the results of validation experiments that assess the accuracy of the analytic model that is developed in this paper. The validations are performed against SimOS, using SimOS' detailed MXS processor simulator and its NUMA memory system simulator. SimOS runs IRIX 5.3, and all benchmark results include OS behavior that occurred while the benchmark was running. Thus, we measure analytic model inputs and estimate system performance for the complete system behavior, instead of for the application alone.

The validation experiments include FFT, LU, Radiosity, and Radix from the Splash-2 suite [12]. Table 6 shows the data sets used for each application.

app .	input size
FFT	-l6 -n1024
LU	512x512 array, 16x16 blocks
Radiosity	-batch
Radix	1M integers, radix 1024

Table 6. Benchmark Data Sets

We attempted to obtain SimOS results for the rest of the Splash-2 benchmark suite, but these applications would not run successfully on the version of the SimOS MXS processor simulator that we were able to obtain access to. Similarly, we were unable to make this version of the SimOS MXS simulator produce results for greater than 8 processors. Although the number of benchmarks that ran successfully is small, the memery access characteristics captured in the model input parameters vary greatly accross these applications as well as in the different periods between barriers in a given application, and thus the analytical model is exercised over a non-trivial region of the input parameter space. Tables 3 and 5 illustrate some of the variety in the memory request behavior accross the applications. As shown below, the processor throughput varies from 0.1 to 2.4 instructions per cycle across the application barriers that we were able to validate against, indicating that the differences in memory request behavior among these benchmarks is significant. The low processor throughput estimates also indicate that although the number of processors is relatively small, significant contention occurs in the memory system (particularly at the directory controllers), and thus the ability of the analytic model to accurately estimate queueing delays is also exercised. This is confirmed by the measured mean queueing delays reported by SimOS for these applications (with the architectural parameters in Table 1).

The validation results for model input parameter values that exhibited the greatest degrees of heterogeneity in processor performance are shown in Figures 6, 7, and 8. These results are for specific barriers (i.e., inter-barrier phases) of FFT, Radix, and Radiosity, running on 8-node and 4-node versions of SimOS. Each graph gives the throughput (in IPC) for each processor estimated by the new heterogeneous sytem model as well as the average throughput estimated by the homogeneous model. Results for other barriers of the FFT, LU, Radix, and Radiosity benchmarks (both 8-node and 4-node) are presented in Table 7. The column numbers in the table correspond to node numbers. For each pair of rows, the first row is the IPC reported by SimOS, and the second row is the IPC predicted by the model. The rightmost column corresponds to a homogeneous model using the average statistics, where the first row is the average IPC across all nodes reported by SimOS, and the second row is the IPC predicted by the previous homogeneous model using input parameters that are averaged across all nodes.

These results show that the analytic estimates of per-processor throughput agree quite closely with the SimOS measurements, even when each processor throughput is remarkably different. The model achieves accurate performance estimates although memory request behavior is modeled statistically and at a high level of abstraction. As mentioned before, the complexity of the new analytic model makes its tractability a key question. In validating the model, however, we discovered no cases where the model did not converge to a solution within a matter of a few seconds, in spite of strong heterogeneity in the model inputs and in the estimated per-processor throughputs.

Although the homogeneous analytic model is also often accurate in estimating the average processor throughput, the new estimates of per-processor throughput from the heterogeneous model are crucial to accurately estimating the impact of the memory system architecture on application execution times, is illustrated in Section 5.2, because for applications employing barriers, the barrier only completes execution when the slowest processor reaches the barrier. Moreover, there are examples, such as LU-4 barrier 4, for which the homogeneous model is not even accurate with respect to the average node behavior reported by SimOS. This is due to the fact that heterogeneity can have non-linear effects on queueing delays in the memory system, and these non-linear effects are only captured in the heterogeneous model.

Validation of the lock synchronization modeling was difficult in that none of the benchmarks that succeeded in running on the MXS simulator in our version of SimOS exhibited significant lock contention. Instead, we implemented the "Sieve of Eratosthenes" algorithm for finding prime numbers, and found that executing this application on SimOS exhibited significant lock contention. The accuracy of the analytic model with complementary service time inflation for lock contention, as compared with SimOS measures for the application, is illustrated by the results in Figures 9 and 10. The application's heavy lock contention is revealed by the high mean lock waiting

app	1	2	3	4	5	6	7	8	average
FFT-4 (b1)	0.12	0.14	0.15	0.17					0.15
	0.11	0.13	0.15	0.17					0.16
FFT-4 (b2)	0.68	0.73	1.06	0.91					0.82
	0.61	0.66	0.99	0.82					0.74
FFT-4 (b3)	0.47	0.48	0.55	0.62					0.52
	0.43	0.43	0.50	0.57					0.47
FFT-8 (b1)	0.14	0.16	0.42	0.61	0.41	0.13	0.33	0.48	0.40
	0.13	0.16	0.43	0.62	0.42	0.13	0.33	0.47	0.40
FFT-8 (b3)	0.52	0.56	0.52	0.53	0.39	0.43	0.47	0.50	0.50
	0.51	0.55	0.55	0.52	0.39	0.42	0.48	0.49	0.50
LU-4 (b2)	1.27	0.59	0.59	1.43					0.82
	1.40	0.61	0.62	1.37					0.91
LU-4 (b3)	2.17	1.79	1.92	2.22					1.96
	2.37	1.88	2.06	2.50					2.13
LU-4 (b4)	0.85	1.09	0.81	1.20					**1.06**
	0.89	1.14	0.72	1.22					**1.32**
Radix-4 (b2)	0.43	0.46	0.63	0.48					0.49
	0.41	0.47	0.58	0.46					0.47
Radix-4 (b6)	0.30	0.36	0.26	0.45					0.36
	0.32	0.39	0.26	0.48					0.39
Radix-8 (b1)	0.64	0.64	0.62	0.64	0.61	0.61	0.61	0.63	0.62
	0.68	0.64	0.61	0.64	0.62	0.61	0.60	0.63	0.62
Radix-8 (b3)	0.90	0.70	0.68	0.64	0.56	0.50	0.42	0.45	0.69
	0.87	0.67	0.67	0.63	0.58	0.52	0.47	0.46	0.69
Radiosity-4 (b9)	0.49	0.78	0.93	0.35					0.63
	0.51	0.76	0.90	0.36					0.65

Table 7. Validation Results: Processor Throughput Estimates

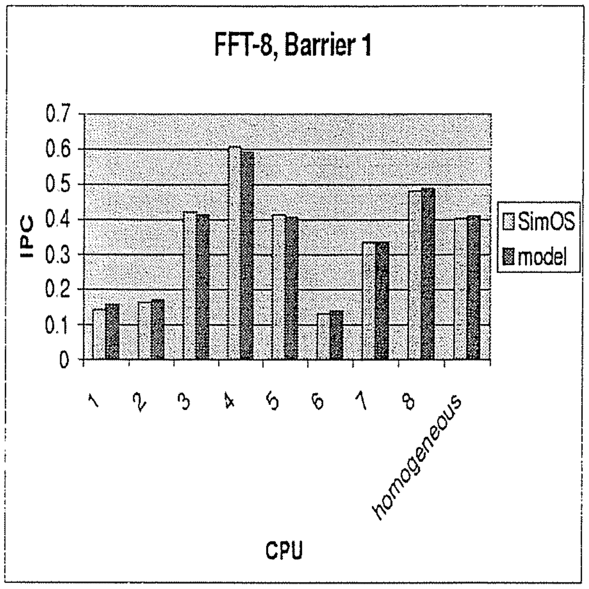

Figure 6. Validation of FFT-8, Barrier 1

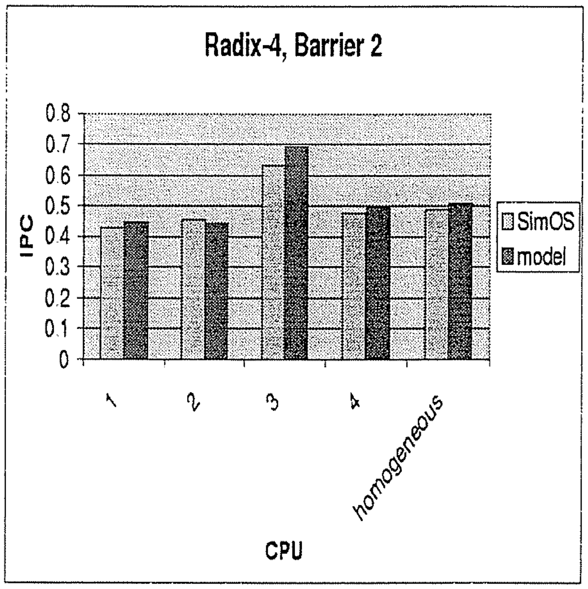

Figure 7. Validation of Radix-4, Barrier 2

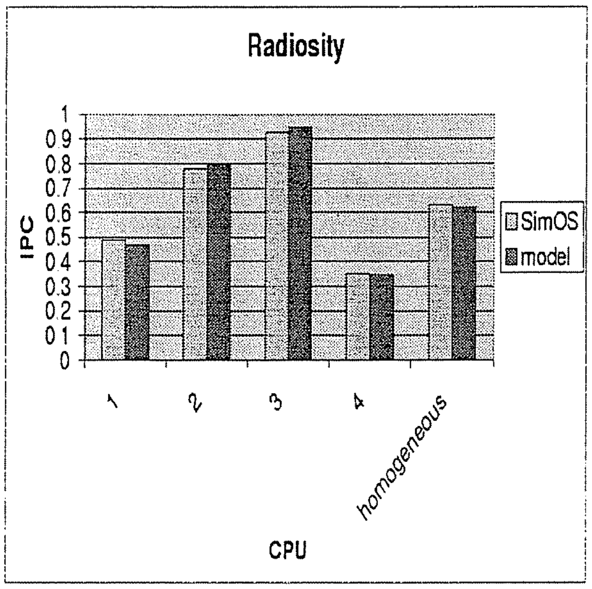

Figure 8. Validation of Radiosity-4

times reported in Figure 9. The mean lock waiting times and processor throughputs (measured in memory requests per cycle) estimated by the analytic model agree reasonably well with the measured values, even in this extreme case.

5.2 Applications of the Model

The model developed in this paper can be applied to efficiently obtain initial answers to numerous questions about architectural designs and the interactions between applications and the architecture. Recall that the memory system parameters (below the L2 cache) can be varied to explore the design space without having to re-run the detailed simulator to obtain new input parameters. The model can reveal performance bottlenecks at specific system resources due to heterogeneous and bursty memory request traffic. This section illustrates three applications of the model to such issues, pointing out cases in which the previous homogeneous model provides inaccurate results.

5.2.1 Decoupling the Network Interface from the Directory Controller

As discussed in Section 2.1, the SimOS architecture requires all traffic into and out of a node to pass through the directory controller (DC). The DC effectively assumes the responsibility of being the network interface (NI) for all traffic, including traffic that does not require use of the directory. This coupling of the DC and NI may, in some cases, create a bottleneck, especially as processor speed increases relative to memory speed.

An interesting architectural question that can be quickly assessed with the model is the performance gain that could be achieved by decoupling the NI from the DC, given that the DC is twice as slow relative to the processor speed as compared to the default values in Table 1. Figure 11 shows that, for an 8-way parallel execution of Radix (barrier 2), decoupling the NI and the DC reduces the cycle count of the barrier from 163k to 124k and moves the bottleneck from CPU 3 to CPU 7.

The rightmost pair of bars shows the performance impact predicted by the homogeneous model, and there are two key inaccuracies worth noting. Quantitatively, the homogeneous model predicts barrier cycle counts 30% less than those predicted by the heterogeneous model. Qualitatively, the homogeneous model fails to capture the shift in the bottleneck.

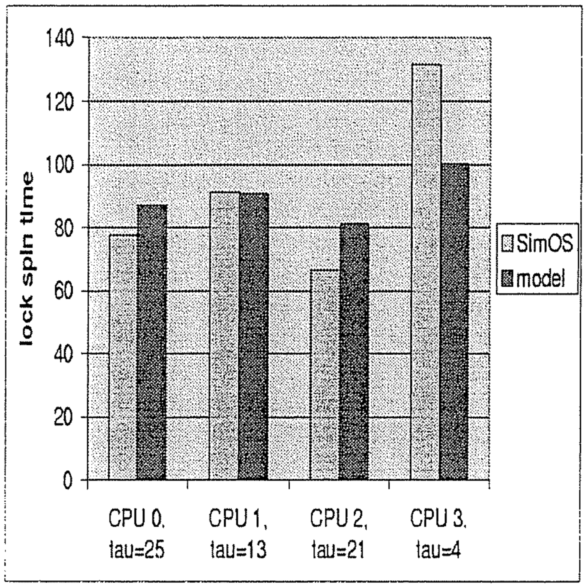

Figure 9. Validation of Lock Contention Estimates ("Sieve of Eratosthenes" algorithm on 4 CPUs)

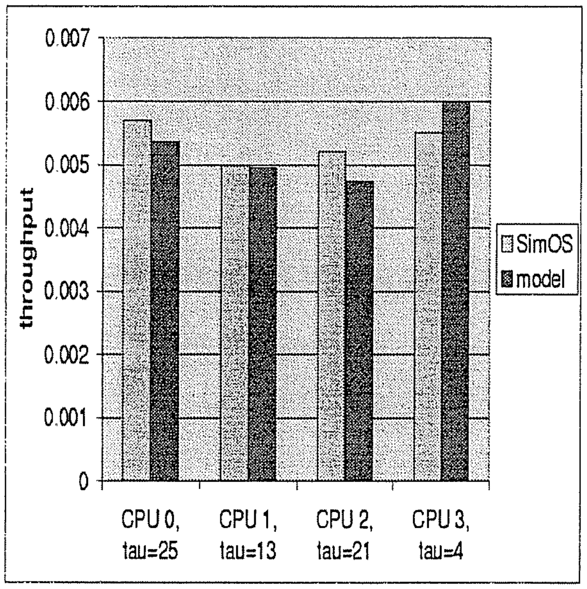

Figure 10. Validation of Processor Throughput Estimates with Lock Contention ("Sieve of Eratosthenes" algorithm on 4 CPUs)

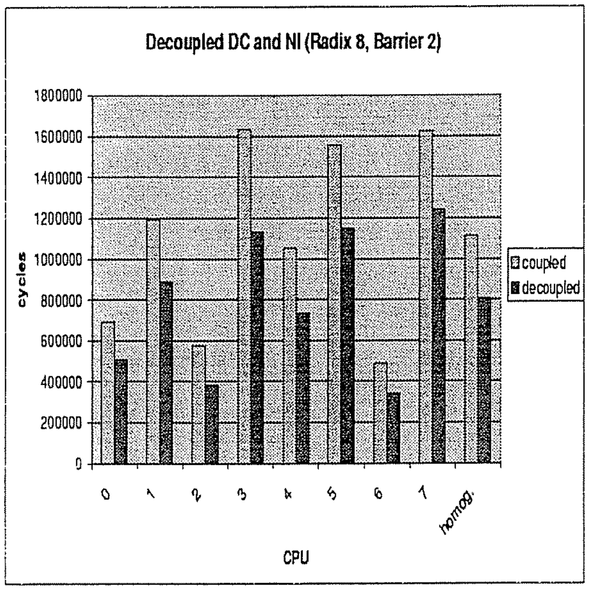

Figure 11.

5.2.2 Importance of Data Layout

The model input parameter values for a given application or workload can be useful in identifying opportunities for tuning the application (or kernel). For example, the input parameters that characterize where memory requests from each processor are directed reveal insight into data layout issues. More specifically, good data layout schemes maximize the fraction of requests that can be serviced locally. That is, for a given application, if one layout scheme has a higher probability of servicing a memory request locally than another scheme, then it will likely have better performance, since local requests have lower latency than remote requests.

For barrier 3 of LU-4, Figure 12 illustrates the performance gains that can be achieved for a hypothetical 50% increase in the probability that a request is satisfied locally. Of interest is the reduction in execution time that is achieved by the hypothetical tuning at the bottleneck processor (i.e., CPU 3 in the figure). The magnitude of the decrease in execution time guides how much effort should be expended in looking for opportunities to increase data locality in the LU code.

As shown in the figure, the homogeneous model predicts a decrease in average processor execution time that is similar in magnitude (although larger in percentage) to the decrease in execution time for the bottleneck processor. However, the decrease in average processor execution time is not generally a reliable estimate for what will occur at the bottleneck center.

5.2.3 Tuning the Operating System

As discussed in Section 3, the operating system causes the memory request heterogeneity that is observed in workloads with applications that are generally considered homogeneous. For example, Table 4 shows that barrier 2 of FFT-4 is heterogeneous in τ because of the kernel. To determine the performance gain that could be achieved by hypothetically tuning the kernel for homogeneity, we compared the performance of this barrier against the performance of the same barrier with homogeneous kernel behavior. The OS parameters at each node are assumed to be equal to their averages across all four nodes. Figure 13 reveals that if the OS can be tuned for greater uniformity in processor usage within the time that the application executes between barriers, rather than on a much coarser timescale, this would lead to a 20% reduction in cycle count for this barrier.

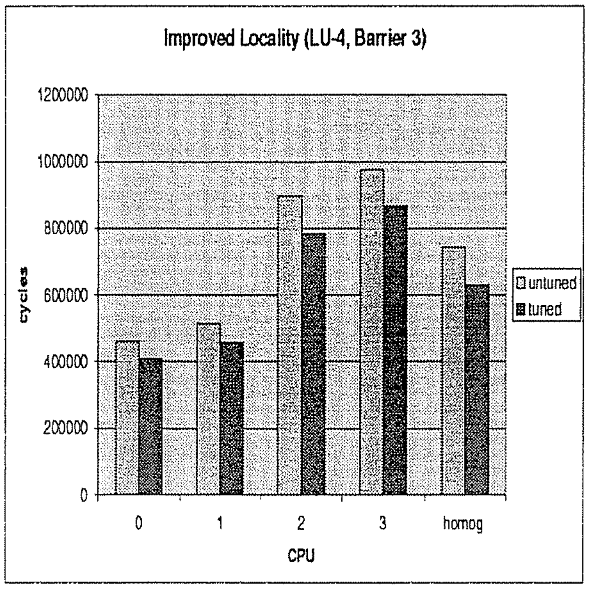

Figure 12. : Performance Impact of Improved Memory Locality

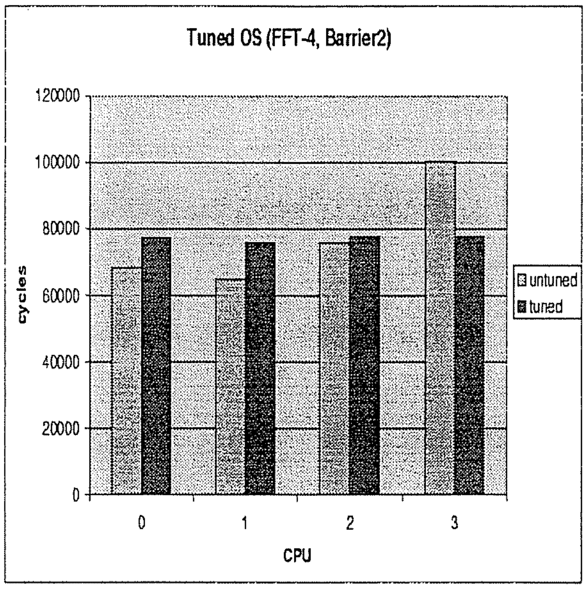

Figure 13. : Predicted Impact of Tuning the OS

6 Conclusions

We have developed and validated a new analytical model for evaluating the performance of shared memory multiprocessors with ILP processors and heterogeneous processor workloads. This work extends prior research in this area in three ways: (1) adapting and validating the model for a different architecture (SimOS) than that considered in previous work, (2) modeling heterogeneous node behavior that was reported by SimOS even when running homogeneous applications, and (3) applying new techniques for modeling bursty memory requests and developing techniques for modeling lock synchronization. Despite the complexity of modeling processor heterogeneity and non-uniform memory access probabilities, bursty memory traffic, and lock contention, the model converges quickly, is still several orders of magnitude faster to solve than detailed simulation, and the number of input parameters remains manageable. The model validates extremely well, for individual processor throughput estimates, over a range of Splash-2 benchmarks that has wide variety in memory request behavior, which leads to a wide range of observed processor throughputs. Examples in setion 5 show how the model can be used to study architectural design issues as well as to study interactions between the architecture and the application. Moreover, the examples show that insight can be gained simply from looking at the input parameter values that are measured for a particular workload.

The model is being made available for use by others in the POEMS environment [2]. The real test of the model is how it performs in a commercial architecture design context, for which public data is unavailable. Perhaps in making the model available commercial systems designers, feedback can be obtained about its accuracy in a real system design setting. Future research topic includes investigating methods of coupling the architectural model with a more abstract model of the communication and synchronization behavior in the application.

References

[1] D. Albonesi and I. Koren. A Mean Value Analysis Multiprocessor Model Incorporating Superscalar Processors and Latency Tolerating Techniques. *Int'l Journal of Parallel Programming*, 1996.

[2] E. Deelman et al. POEMS: End-to-end Performance Design of Large Parallel Adaptive Computation Systems. In *Proceedings of the 1st Int'l Workshop on Software and Performance*, Oct. 1998.

[3] D. Eager, D. Sorin, and M. Vernon. Analytic Modeling of Burstiness Using Approximate MVA. Technical Report 1391b, Computer Sciences Dept., Univ. of Wisconsin - Madison, Dec. 1998.

[4] D. Kroft. Lockup-Free Instruction Fetch/Prefetch Cache Organization. In *Proc. 8th Int'l Symp. on Computer Architecture*, pages 81–87, May 1981.

[5] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proc. 21st Int'l Symp. on Computer Architecture*. Apr. 1994.

[6] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[7] V. Pai, P. Ranganathan, and S. Adve. RSIM Reference Manual. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, Aug. 1997.

[8] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete Computer Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology*, 1995.

[9] D. Sorin et al. A Customized MVA Model for ILP Multiprocessors. Technical Report 1369, Computer Sciences Dept., Univ. of Wisconsin - Madison, Mar. 1998.

[10] D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood. Analytic Evaluation of Shared-Memory Parallel Systems with ILP Processors. In *Proc. 25th Int'l Symp. on Computer Architecture*, June 1998.

[11] D. Willick and D. Eager. An Analytical Model of Multistage Interconnection Networks. In *Proc. ACM SIGMETRICS*, pages 192–202. May 1990.

[12] S. Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. 22nd Int'l Symp. on Computer Architecture*, pages 24–36, June 1995.

[13] XXXX. A Customized MVA Model for Shared-Memory Systems with Heterogeneous Applications. Technical report, 1999.

A Customized AMVA Equations for the Homogeneous Model

Notation: For readability, we have adopted the following subscripts and superscripts for the variables in the model. The resource is always the first subscript on a variable, whether it is mean residence time (R), mean waiting time (W), mean utilization (U), or mean service time (S). For example, R_{dc} is the mean residence time at the directory controller (DC). For many terms, there is a subscript of *loc* or *rem* to indicate whether the resource is at the local node for a given processor or a remote node. The subscript variable y denotes the transaction type (such as read or write).

We now present a subset of the equations used in the homogeneous model. These equations apply to both submodels that are used to compute the two types of stall time: stalling due to load misses awaiting data and stalling due to the hardware constraint on the number of outstanding memory requests. The following equation is for the total response time of a customer through the system. It includes the response times at the processor, bus (both local and remote), network, and directory controller.

$$R = R_{pe} + R_{bus} + R_{net} + R_{dc}$$

Each of these terms is derived from lower level equations. For example, the residence time at the DC is equal to the sum of the residence time at the local DC and at the remote DCs.

$$R_{dc} = R_{dc_{loc}} + R_{dc_{rem}}$$

The mean residence time at the local (remote) DC is equal to the weighted average of the residence time of a transaction type y at the local (remote) DC, for all transaction types y.

$$R_{dc_{loc}} = \sum_y R_{dc_{loc,y}}$$

$$R_{dc_{rem}} = \sum_y R_{dc_{rem,y}}$$

The mean residence time of a transaction of type y at the local (remote) DC is equal to the probability of transaction y times how many times it visits the local (remote) DC ($V_{dir_{loc_y}}$) times the sum of the waiting time at the local (remote) DC ($W_{dc_{loc}}$) and the service time at a DC.

$$R_{dc_{loc,y}} = P_y V_{dc_{loc_y}} (W_{dc_{loc}} + S_{dc})$$

$$R_{dc_{rem,y}} = P_y V_{dc_{rem_y}} (W_{dc_{rem}} + S_{dc})$$

All of the terms in the above equations are inputs except the waiting times. $W_{dc_{loc}}$ consists of the waiting time at the local DC due to requests from the local node ($W^{loc}_{dc_{loc}}$) and due to requests from remote nodes ($W^{rem}_{dc_{loc}}$).

$$W_{dc_{loc}} = W^{loc}_{dc_{loc}} + W^{rem}_{dc_{loc}}$$

The waiting time at the local DC due to requests from the local node equals the sum of the waiting times over all transaction types y that cause waiting.

$$W^{loc}_{dc_{loc}} = \sum_y W^{loc.y}_{dc_{loc}}$$

$$W^{rem}_{dc_{loc}} = \sum_y W^{rem,y}_{dc_{loc}}$$

$W_{dc_{rem}}$ consists of the waiting time due to remote customers that are not from that remote node ($W^{others}_{dc_{rem}}$) and those that are from that remote node ($W^{rem}_{dir_{rem}}$).

$$W_{dc_{rem}} = W^{others}_{dc_{rem}} + W^{rem}_{dc_{rem}}$$

$$W^{others}_{dc_{rem}} = \sum_y W^{others,y}_{dc_{rem}}$$

$W_{dc_{rem}}^{rem} = \sum_y W_{dc_{rem}}^{rem.y}$

The following equations are for the mean waiting times due to waiting for specific transaction types. For example. $W_{dc_{loc}}^{loc.y}$ is the waiting time at the local DC due to local requests of transaction type y. Mean waiting time for a single other customer equals $\frac{R_{dc_{loc.y}}}{R} - U_{dc_{loc.y}}$ (the probability that a customer is in the queue but not in service) times the service time, plus $U_{dc_{loc.y}}$ (the probability that a customer is in service) times the mean residual life of a customer in service. Therefore, to get the total mean waiting time, we multiply by the number of local customers who could cause an arriving local customer to wait, $M - 1$.

$W_{dc_{loc}}^{loc.y} = (M-1)\left[\left(\frac{R_{dc_{loc.y}}}{R} - U_{dc_{loc.y}}\right) S_{dc} + U_{dc_{loc.y}}\left(\frac{S_{dc}}{2}\right)\right]$

$W_{dc_{loc}}^{rem.y} = M\left[\left(\frac{R_{dc_{rem.y}}}{R} - U_{dc_{rem.y}}\right) S_{dc} + U_{dc_{rem.y}}\left(\frac{S_{dc}}{2}\right)\right]$

$W_{dc_{rem}}^{others.y} = [(M-1) + M(N-2)][(\frac{R_{dc_{rem.y}}}{R} - U_{dc_{rem.y}})S_{dc} + (U_{dc_{rem.y}})(\frac{S_{dc}}{2})]$

$W_{dc_{rem}}^{rem.y} = M[(\frac{R_{dc_{loc.y}}}{R} - U_{dc_{loc.y}})S_{dc} + (U_{dc_{loc.y}})(\frac{S_{dc}}{2})]$

Lastly, we have the utilization equations. The first equation is the mean utilization of a DC by a local customer, and the second equation is the mean utilization of a DC by a remote customer.

$$U_{dc_{loc.y}} = \frac{P_y}{R}(V_{dc_{loc.y}} S_{dc})$$

$$U_{dc_{rem,y}} = \frac{P_y}{R}(V_{dc_{rem,y}} S_{dc})$$