

# A Processor Queuing Simulation Model for Multiprocessor System Performance Analysis

Thin-Fong Tsuei and Wayne Yamamoto

Sun Microsystems, Inc.

{thin-fong.tsuei,wayne.yamamoto}@sun.com

## Abstract

*This paper describes a processor queueing simulation model for a complex processor that aggressively issues instructions with the use of out-of-order, multiple issue and multithreading. The model is developed for memory and system evaluation of memory-intensive commercial OLTP (On-line Transaction Processing) workloads on large multiprocessor systems. Our approach differs from detailed cycle accurate and direct-execution simulations in that we do not simulate instruction execution. Instead, we take a high level view, as in analytical models, and model a minimal set of processor components to accurately generate the memory access traffic for system simulation. The model is validated with a cycle accurate simulator and is within 5% accuracy. Results on the effect of store burstiness and memory latencies on overlapping of cache misses and buffer sizing are presented.*

## 1. Introduction

The performance requirements of many commercial applications can only be met with large, complex multiprocessor systems. Modeling and performance analysis for large systems are becoming increasingly important. With processors becoming more complex through the use of out-of-order multiple issue, multithreading, and chip multiprocessing, multiprocessor system models based on simple processor models no longer accurately represent the memory access workload to other system components. In particular, simple serial issue, block-on-cache-miss processor models cannot capture the characteristics of overlapping cache

misses and processor stalls due to resource contention.

Cycle accurate processor simulators which simulate the micro-architecture of processors are essential and commonly used for research and design of processors [5]. However, this type of model, while accurate, is too complex and slow to be used effectively in large multiprocessor system models. Techniques based on direct-execution were developed to speedup execution and reduce complexity in processor simulations [2,3,4,8]. Since direct-execution simulation runs application code directly on a host machine, it may not be feasible for commercial applications that can easily require configurations of gigabytes of memory and disk space.

In analytical multiprocessor models, processors are generally modeled as a blackbox with emphasis on defining parameters that effectively characterizing the memory access arrival process. Parameter values are derived from measurements on existing system or obtained from other simulations. Very often, synthetic workloads are used to stress the system components to identify bottlenecks.

In this paper, we describe a processor queueing model for multiprocessor system performance analysis. The purpose of this processor model is for memory hierarchy and system design evaluation of memory-intensive commercial OLTP (On-line Transaction Processing) workloads on large multiprocessor systems. Our approach differs from detailed cycle accurate and direct-execution simulations in that we do not simulate instruction execution. Instead, as in analytical models, we view the function of a processor model is to provide the interactions between the processor and the rest of the system. We model a minimal set of pro-

---

Sun, Sun Microsystems, and the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

processor characteristics that accurately generate the memory access workload of a complex processor.

We note that a processor stalls due to contention for processor buffer resources in the cache hierarchy and other application characteristics in the processor pipeline. These buffer stalls are heavily dependent on the memory operations of the application and the system latencies. Consequently, we decouple the processor memory subsystem buffers from other pipeline operations. Our model treats the core pipeline as a black box and focuses on modeling the interactions among core pipeline and buffers in the processors memory hierarchy.

The processor queueing model is extensively validated with cycle-accurate processor simulation. While previous studies mainly validated using scientific workloads, we focus on commercial OLTP applications, and use the TPC-C (Transaction Processing Council) benchmark [13] with commercial database software as a workload. In the process of validating the model, we found that the model was sufficiently accurate for analyzing trade-offs in sizes and allocation policies for the buffers in the processor's memory subsystem.

We use the processor queuing model to investigate how stall cycles and the overlapping of cache misses vary with memory latencies. Pai et. al. [6] have found that read clustering, or burstiness, is required to achieve high overlapping on memory latency for SPLASH workload. Our results show that the stores of the OLTP workload are bursty. In addition, though bursty loads and stores improved the overlapping of memory access time, they are also more likely to fill up the buffers and cause the processor to stall. Proper sizing of the buffers and the cache architecture are important to reduce such processor stall time

The rest of the paper is organized as follows. Section 2 discusses some of the related work. Section 3 describes the model. Sections 4 and 5 present the validation and overlap study results. Section 6 discusses integration of the processor queueing model with the system model. Section 7 concludes the paper.

## 2. Related Work

There are several approaches to multiprocessor performance system evaluation. The most conservative approach is to do cycle-by-cycle simulation for the entire system from the processor to the interconnection network and the memory system [5,7]. This approach is detailed and has good accuracy, but is

extremely slow. It is unpractical for evaluating a large multiprocessor system.

Direct execution is an alternate system simulation method with improved simulation time over the cycle accurate simulation [2,8]. As with the cycle accurate simulation, direct execution also models the actual execution of the instructions of an application on a processor. Direct execution, however, decouples the processing of the functional and timing components of instruction execution. The functional simulation is accelerated by executing the binary of an application directly on a host machine. The timing information is analyzed independently. Pai and Adve have done extensive studies with Ranganahan [6] and Durbhakula [2] on issues and methods to improve the accuracy of direct execution. Direct execution has improved the simulation time by a few times, but it still simulates many micro-architecture details. The size of the system that it can be applied to is limited.

Analytical modeling using mean value analysis (MVA) has been used in shared memory multiprocessor systems with processors that block on cache misses [15]. This technique is shown to be efficient and reasonably accurate for large systems. The new challenges in analytical modeling for today's superscalar, out-of-order processors are to model the overlap in memory accesses on cache misses and to estimate processor stall time. New parameters, such as fraction of overlap in load misses, number of outstanding misses, and probability of blocking due to full buffers, are defined to model the new memory access characteristics [1,10,14]. Except for [10], synthetic workloads are used for parameter values. In [10], the application parameters are obtained from a simplified processor simulation derived from DirectRSIM [2], including the blocking probabilities for a single buffer. Though the simulation is simplified and has achieved two orders of magnitude speedup over RSIM [5], it maintains the complexity needed to simulate instruction execution.

All the analytical studies assume that overlap and blocking parameter inputs are static and are unaffected by the contention delay at the rest of the system. In practice, blocking due to buffer capacity is a function of the latency and the contention delay at the system, and iterations between the processor simulator and the MVA system model are needed in high contention conditions. Our model can be used to derive the blocking parameter values quickly, and is much more

lightweight to iterate since we do not need to execute the entire application.

### 3. The Processor Model

#### 3.1 Processor overview

The processor modeled in this study was designed to exploit both instruction level and thread level parallelism while running at a high clock frequency. It executes the SPARC™ instruction set and features deep pipelining with superscalar, out-of-order issue, speculative execution, and 2-way multithreading. Capable of fetching multiple instructions per cycle, the instruction fetch unit involves a two level adaptive branch outcome predictor and a branch target predictor. Fetched instructions are decoded, renamed and inserted into an instruction issue buffer (IIB). Instruction fetching is stalled once the IIB entries are filled. Once in the IIB, an instruction can be issued as soon as all its operands are available. Multiple instructions can be dispatched to functional units every cycle.

The cache hierarchy consists of three levels of cache: separate L1 instruction and data caches, a unified L2 cache, and a unified L3 cache. Loads and instructions that miss in the L1 caches are moved into a L2 load buffer (LB) which can process instruction/data read misses concurrently. When the LB fills up, processing of L1 cache misses is blocked and will eventually stall the processor. Read misses that miss in the L2 and L3 cache are sent out of the processor over a split transaction interconnect to the memory controllers.

A store remains in the IIB until it is the oldest instruction, then it is moved into the store buffer (SB). Once in the SB, stores are committed to the L2 cache in program order. Store misses requiring memory access can cause the SB to fill up which will back up into the IIB and may eventually cause the processor to stall.

The processor is highly speculative and uses both control and data speculation to boost performance. We use aggressive control speculation employing the branch predictor to fetch and execute instructions beyond multiple unresolved branches. On a misspeculated branch, instructions younger than the

branch are flushed from the IIB and LB and instruction fetching resumes along the correct path. Data speculation is also employed to speculate on the value of loads. Correctly speculating on a load allows earlier dispatch of dependent instructions, misspeculation results in reissuing of these instructions.

#### 3.2 Model overview

As discussed earlier, the goal of the processor model is to accurately reproduce the memory access traffic, that is, the memory access types and the rates at which they are generated.

As the instruction fetch has well defined and less complex blocking behavior, we decouple the processing of buffer management from the instruction fetching and core pipeline (Figure 1). In the processor modeled, as long as there is no mispredicted branch or instruction cache miss and the IIB is not full, instructions are fetched and installed into the IIB. Since we are mainly concerned with memory hierarchy and system design studies from L1 cache outward, we simplify the model by aggregating the fetch time with other pipeline operations and L1 cache access time into one service demand which feeds the IIB. This service time is the average execution time in an ideal L2 cache environment, including pipeline interlocks due to data dependencies, L1 cache misses, and branch mispredictions.

The contention for caches and buffer resources in the model adds delays and reduces the instruction execution rate. We model the instruction, load, and store misses as they travel between the buffers (IIB, LB, and SB), capturing the contention and

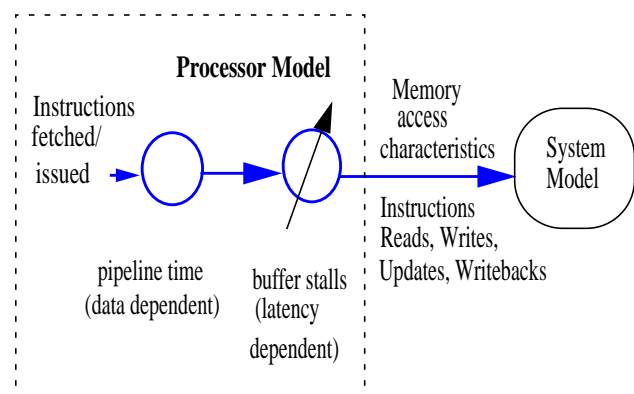


Figure 1. Breakdown of execution time in a processor

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

Table 1: Model Parameters

Application parameters	Measurement of parameter value
Service demand at the pipeline	Derived from cycle-accurate processor simulation with infinite L2 cache
Histogram of instructions per fetch	Derived from cycle-accurate processor simulation with infinite L2 cache
Loads per instruction Stores per instruction	Statistics collected from trace analysis
Histogram of consecutive store	Statistics collected from trace analysis
Cache misses: Level 1, 2 and 3 instructions, data reads, data writes, upgrades and dirty writebacks	Statistics collected from cache simulations and cycle-accurate processor simulation
<b>System parameters</b>	
Cache and memory latencies	Design specifications, values vary
Buffer sizes	Design specifications, values vary

eventual stalls. The contention delay for buffers is determined largely by cache miss rates and the memory latencies, the service time is relatively independent of the memory latencies.

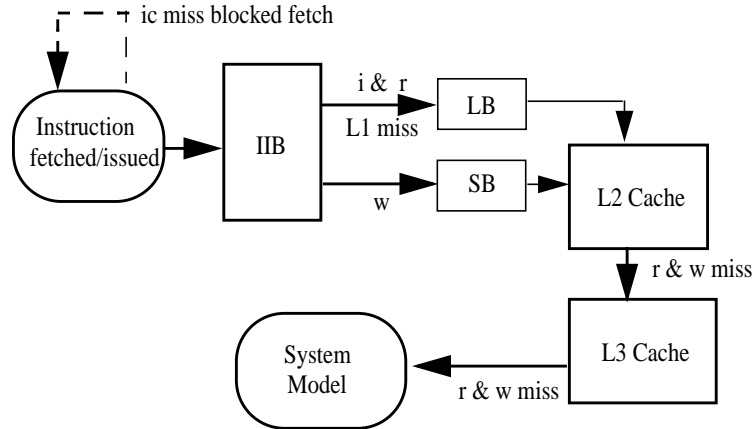
Table 1 defines the model parameters. The service time for the core pipeline is the time to complete the instructions in a fetch when the L2 cache is infinite. The number of instructions in each fetch is determined by the application basic block characteristics and the processor fetch unit architecture. We use a histogram for the number of instructions per fetch. Other application characteristics include loads and stores per instruction, instruction fetch and data load and store cache misses per instruction, and the upgrades and writebacks per instruction. The histogram of consecutive stores can be used to model the store traffic more accurately

Figure 2 details the queueing model for the processor. The service demand for the core pipeline is modeled with two identical delay servers, one for each thread. At the end of the service time,  $n$  instructions are generated according to the instructions per fetch histogram. Every instruction resides in the IIB until it retires. Load and store instructions are assigned based on their per instruction probabilities. Instruction fetch and data cache misses are determined randomly according to their respective miss rate. On an instruction miss, the fetch process stalls until the missed

instruction returns. A data load miss proceeds to the load buffer (LB) and proceeds to L3 cache and the system memory model when it misses in L2 and L3 caches respectively. For validation of the processor, we treat the system model as a delay server, i.e., a constant memory latency.

The in-order retirement of instructions is enforced by the IIB. If an instruction completes before older instructions, it remains in the IIB until it becomes the oldest. Stores remain in the IIB until they are the oldest and then they are moved to the SB where they must commit in order. Load misses are assumed to be independent and complete (i.e., can be removed from the LB) as soon as the data is returned. In general, load misses can be dependent. However, results of internal studies using an infinite buffer size [11,12] indicate that the effect of load-load dependencies on execution rate is low for OLTP workloads because of the random nature of the data accesses. For workloads where this effect is significant, the model can be modified to add a probabilistic measure of load-load dependencies.

While the highly speculative nature of the processor helps to boost performance, misspeculation results in more memory traffic due to cache pollution and possibly more stalls due to LB buffer contention. In our model, we account for misspeculation through a higher cache miss rate obtained from the detailed



**Figure 2. Processor Model**

cycle accurate simulations. We do not model the LB and IIB flushing of the wrong path instructions on a branch misprediction since we compared the miss rates with and without branch misprediction and discovered that the differences were small. However, the model can be refined to flush the buffers with a branch misprediction probability.

The implementation of the model is relatively straightforward and is implemented using a commercial simulation package. The buffers are implemented as pools of numbered tokens. Tokens are retrieved from a buffer pool in sequential order and are put back whenever a buffer space is released. A store can be removed from the IIB once it is the oldest and acquires a space in the store buffer. When LB or SB is full, IIB can become congested and instruction fetches will eventually stall when the IIB is full.

We use two service centers to implement the pipeline cache. The first service center models the first stage of the pipeline. The second service center is simply a delay in the rest of the cache pipeline depth. We model alternate load and store queue scheduling, and first-come-first serve scheduling at the caches.

### 3.3 Obtaining Model Parameter Values

Table 1 also summarizes the measurement and estimation of the model's parameter values. The pipeline service demand is derived from the cycles per instruction when the L2 cache is infinitely large. This parameter and other application parameter values, such as the histogram of instructions per fetch and the

loads and stores per instruction are derived from a single run of the detailed simulation model. An additional run is needed to derive the pipeline service demand for dual-threaded execution. The cache miss rates are obtained from cycle accurate simulations but could also be obtained using simple cache simulations. Both single and dual threaded cache simulations are run. Cache miss rates can also be varied independently for sensitivity studies. The histogram of consecutive stores was obtained empirically and was used to control the burstiness of stores. System parameters, such as buffer sizes, are obtained from processor design specifications.

The model application parameters are defined so that we can leverage on data from detailed trace-driven processor and cache simulation models that are always created in the development of new architectures. The histogram of instructions per fetch, and the values of load and store per instruction can also be estimated from other tools when a detailed simulator is not available. One such suite is the data prefetch analysis tools used in compiler technology [12]. These tools model the fetch unit and the branch prediction unit for data dependency analysis. The service demand can be estimated by static analysis, from older cycle accurate simulators or from existing machines.

Table 2: Processor Parameters

Instruction Issue Buffer	128 entries
L2 Load Buffer	32 entries
Store Buffer	32 entries
L2 cache	1MB, 4-way set associative
L3 cache	32 MB, direct mapped

## 4. Model Validation

### 4.1 Validation with Cycle Accurate Simulation

The model is validated with a cycle accurate processor simulation. The cycle accurate simulator models the processor and system architecture in fine detail. The simulator is trace driven and simulates execution on a cycle-by-cycle basis. The processor and system parameters were chosen to reflect typical systems in the near future. Table 2 summarizes the baseline processor parameters used in our validation. We use a constant 400 cycle memory latency for the baseline configuration.

For validation, we examined the IPC (Instructions Per Cycle), histograms of the buffers usage, and histograms of the outstanding memory loads and stores. We compare the results for two OLTP traces, Trace-A and Trace-B, collected from commercial

Table 3: Comparison of IPC and number of outstanding Loads and Stores

	Single Thread Queueing/ Cycle Accurate	Dual Thread Queueing/ Cycle Accurate
<b>IPC</b>		
Trace-A	-0.43%	0.05%
Trace-B	9.03%	9.51%
<b>Number of Outstanding at the System (Trace-B)</b>		
Loads	2%	0.61%
Stores	3.2%	4%

database software. Trace-A is from a small configuration and Trace-B is from a larger configuration.

Table 3 shows the percent differences in IPC compared to the cycle accurate simulation for single and dual thread. The estimated IPC agreed very well for Trace-A for both single and dual thread. The queueing model estimates is about 10% more optimistic for Trace-B. The main differences between the two traces are the cache miss rates. The cache miss rates of Trace-B are significantly higher (> 50% more). To investigate the causes of the estimated differences, we studied how the two models compared in estimating the overlapping of the load and store misses. The findings for the dual thread results for Trace-B, which have the highest discrepancy between the two models, are discussed. The percent difference of IPC improves to 4.4% for dual thread Trace-B after we added store burstiness to the queueing model.

Table 4: Distribution of Loads and Stores at System for Dual Strand (trace-B)

Number at System	Loads (percentages)		Stores (percentages)	
	Cycle Accurate	Queueing	Cycle Accurate	Queueing
1	61.1	48.3	19.5	10.7
2	23.3	38.2	22.1	19.9
3	10.7	11.4	19.1	23.5
4	3.5	1.9	14.8	20.3
5 and more	1.4	0.2	13.5	26.8

Comparison of the number of outstanding loads and stores at the system (Table 3) for Trace-B showed that the queuing model is more optimistic. More loads and stores are able to access the memory concurrently and, hence, benefit more from overlapping. Table 4 shows the distribution of the number of cycles at which there are 1 to 5 and more number of outstanding loads and stores at the system. For loads, the queuing model has greater discrepancies at 1 and 2 loads, and is slightly optimistic in load overlapping. For stores, higher clustering when there are larger numbers of outstanding stores indicates that the queuing model is not blocking store misses often enough.

We further investigate the buffer usage. We find that the differences in the utilization of the buffers, as shown in Table 5, are within 6%. The utilization of SB is much higher than LB because we model a L1 write through cache. Nonetheless, the stores have a higher miss rate than loads at the L2 cache. Figures 3, 4 and 5 compare the buffer usage distribution for IIB, LB, and SB respectively. The IIB usage in Figure 3 shows that the two models agree well in the lower range but diverge at the higher end. The lower probability that the IIB is full means that the queuing model is less likely to be blocked.

Figure 4 shows that the queuing model has a higher LB usage at the lower range. In the queuing model, we omitted the effect of aliasing on multiple occupancy in the load buffers; as a result, the LB is less likely to have high concurrent occupancy than the cycle accurate model indicates. However, because the processor queuing model has higher instruction execution rate, more load misses arrive randomly at the LB. This explains the differences at a lower occupancy of LB and a higher average number of LB used. Since the usage of the load buffer is very low and a

Table 5: Buffer Utilizations for Dual Strand (trace-B)

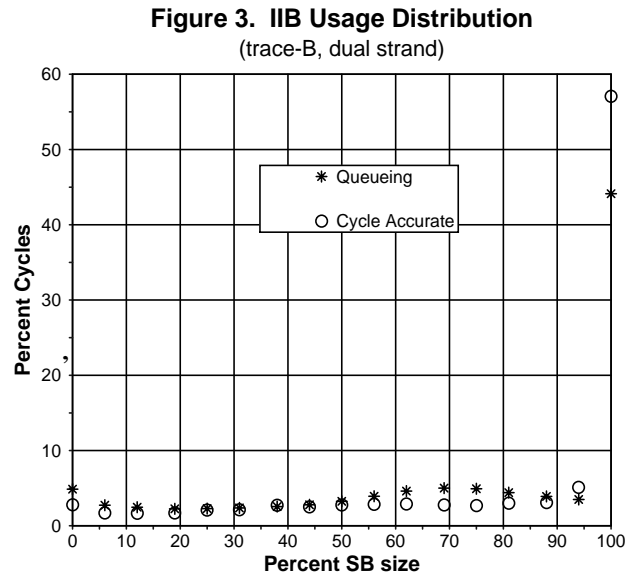
Buffer Type	Dual Strand (trace-B)	
	Cycle Accurate	Queueing
IIB	39.6%	35.9%
LB	12.2%	17.2%
SB	44.3%	38.9%

more refined model would have negligible improvement on the accuracy of the model, thus, we chose to keep the simple load buffer model.

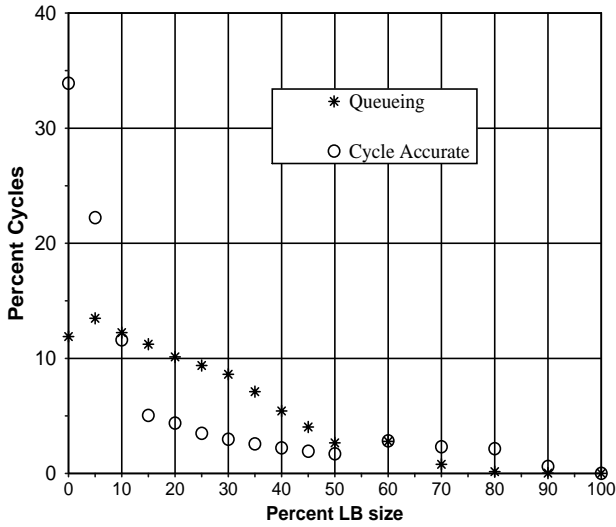
The store buffer usage in Figure 5 shows that the cycle accurate simulator has a burstier arrival of stores. There is a higher peak when the buffer is full. The queuing model, on the other hand, is more uniformly distributed at higher occupancy. This is expected since the loads and stores are uniformly distributed in the instruction stream for the queuing model. We will discuss the effect of burstiness of stores in the sections 4.2 and 4.3 below.

## 4.2 Sensitivity Studies

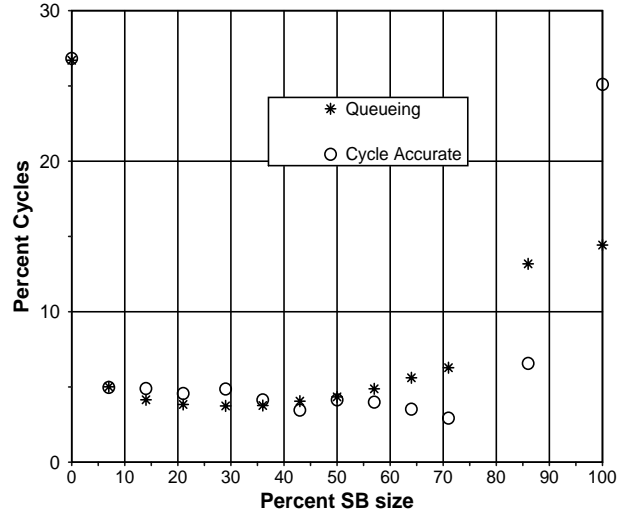
We did sensitivity studies by varying the cache sizes and the store and instruction buffer sizes. The LB size is unchanged because its usage is low and has little effect. The L2 and L3 cache sizes are doubled, and the store and instruction buffer sizes are increased by two and four times. Table 6 compares the percent gains in IPC (Instructions per Cycle) predicted by the cycle accurate simulator with those from the queuing model. It shows that at the base cache and store buffer sizes, the difference in percent gain from the two



**Figure 4. LB Usage Distribution**  
(trace-B, dual strand)



**Figure 5. SB Usage Distribution**  
(trace-B, dual strand)



models is the largest. The cycle accurate model gains little by doubling the IIB size, while the queueing model gains a 16%.

As the contention on the store buffers is alleviated by either increasing the cache sizes and, hence, lowering the store cache miss rates or by doubling the store buffer size, the performance gain from architecture changes estimated from the two models are similar. The same design decision would have been made using either model.

From these results, we believe that the queueing model is not capturing the store buffer pressure as in the cycle accurate model. This difference becomes significant as more instructions are fetched when IIB is doubled. It is evident that the burstiness of store arrivals is important in identifying the correct size for the store buffer, which has significant impact on the performance.

*Table 6: Sensitivity Studies Results*

<b>L2 (MB)</b>	<b>L3 (MB)</b>	<b>SB</b>	<b>IIB</b>	<b>%IPC gain Cycle Accurate</b>	<b>%IPC gain Queueing</b>
1	16	32	64	base	base
1	16	32	128	5.9%	16.2%
1	16	64	128	11.8%	18.3%
1	32	32	64	2.9%	5.5%
2	16	32	64	14.7%	11.6%
2	16	32	128	23.5%	26.4%
2	16	64	128	26.5%	28.2%

Table 7: Distribution of Consecutive Stores

Number of consecutive stores	Random Assignment (%)	Burstiness Histogram (%)
1	68.1	41.3
2	24.9	28.7
3	6.0	0.0
4	0.9	0.0
5	0.1	0.0
6	0.0	0.0
7	0.0	16.5
8	0.0	13.5

### 4.3 Burstiness of Stores

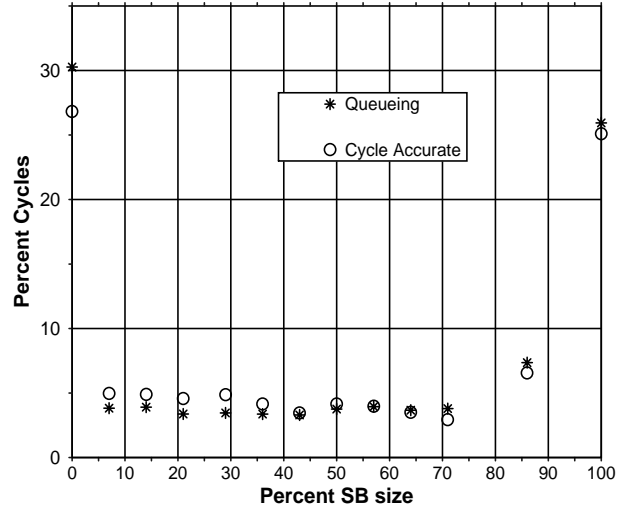
To verify that the burstiness in store arrivals is the reason that store buffer in the queueing model is not blocked as frequently as in the cycle accurate model, we modify the random assignment of stores in the queueing model to use a histogram to generate consecutive stores in each fetch. The average number of stores per instruction and the store miss per instruction remained unchanged.

Table 7 shows the distribution of consecutive stores in each fetch with random assignment and using an input histogram. Figure 6 shows that the store buffer usage of the two models agreed well. The difference in IPC and the average number of busy store buffers between the two models improve by 5% for the base configuration as shown in Table 8.

## 5. Overlap (Hidden) Miss Penalty Study

In an ideal buffer for an out-of-order, superscalar processor, one expects the longer the latencies, the more concurrent misses and greater the overlapping (or hiding) of miss memory access times. However, in practice, there are limited buffers. As latencies increase, buffers are held longer and the processor eventually stalls waiting for the buffers. After validating the queuing model with the cycle accurate model for different buffer sizes and latencies, we vary the memory latency to study the performance benefit of overlapping of misses as the latency increases, or alternatively, how to size the buffers with the memory latency.

Figure 6. SB Usage Distribution with Bursty Stores (trace-B, dual strand)



To measure the benefit of overlap, we compare the hidden miss penalty (%Overlap) to that in a block-on-miss processor. To calculate %Overlap, we define the Load Miss Penalty as the number of cycles to resolve all load misses and Cycles Blocked as the number of cycles the processor was blocked due to memory operations. The %Overlap is the fraction of the Load Miss Penalty that are overlapped, i.e., the percent difference between the Load Miss Penalty and Cycles Blocked. In a blocking processor (i.e., the processor stalls when there is a load miss) this overlap factor would be equal to zero.

Table 8: Comparison of IPC and SB Usage with Store Briskness

	Random Assignment	Burstiness Histogram
<b>IPC</b>		
Cycle Accurate/ Queueing	+9.5%	+4.4%
<b>SB Usage</b>	38.9%	42.7% %

**Table 9: Processor Stall and Cache Miss Overlap for a Range of Latencies**

Latency processor cycles	%Cycles Blocked	%Overlap (i+r)	%Cycles SB is Full
200	<b>34.9</b>	<b>61.7</b>	<b>11.8</b>
330	<b>45.8</b>	<b>47.8</b>	<b>28.8</b>
460	<b>55.7</b>	<b>31.5</b>	<b>38.6</b>
730	<b>69.0</b>	<b>4.8</b>	<b>48.2</b>
1000	<b>76.2</b>	<b>-11.3</b>	<b>52.5</b>

Table 9 shows the %Overlap factor as the memory latency in processor cycles is varied from 200 to 1000 cycles using the baseline buffer sizes and cache configuration. The results show that with a 200 cycle latency, as much as 61.7% of load miss latency is hidden. However, as the latencies increase, the %Overlap actually decreases. This is due to the increase in the %Cycles Blocked. From the results, we can see that the %Cycles Blocked doubles for a five fold increase in latency. Further examination revealed that the increase in %Cycles Blocked as the latency becomes longer is due to the SB being full. Table 9 shows the %Cycles the SB is full increases as the latency increases. In fact, as the latency increases, the blocking due to the SB full eventually results in the %Overlap decreasing until it finally disappears when the store buffer is about 50% of time full.

We did the same set of runs with one sixth store miss rate to reduce the pressure on the store buffer. The results shown in Table 10 show a definite decrease in %Cycles the SB is full as the latency increases. For a 200 cycle latency, stores have little effect and the percent of load misses that overlap is about the same as our previous result. For latencies greater than 200 cycles, the %Overlap is much better than our previous case. This shows that for the larger latencies, stores can have a significant effect on the %Overlap factor.

**Table 10: Processor Stall and Cache Miss Overlap at a lower store miss rate**

Latency processor cycles	%Cycles Blocked	%Overlap (i+r)	%Cycles SB is Full
200	<b>33.5</b>	<b>62.7</b>	<b>3.3</b>
330	<b>41.5</b>	<b>54.1</b>	<b>8.4</b>
460	<b>48.8</b>	<b>45.3</b>	<b>12.6</b>
730	<b>60.5</b>	<b>30.3</b>	<b>18.5</b>
1000	<b>67.6</b>	<b>21.0</b>	<b>22.1</b>

## 6. Integration with System Model

We are currently investigating how to integrate the processor queuing model into multiprocessor system performance evaluation. One approach is to link the processor queuing model with a system model for an end-to-end simulation, as is done conventionally. Another approach is to iterate between the processor queuing model and a system model. This is particularly useful when the two models are developed using different tools, or of different types (e.g., an MVA system model).

In the analytical model described in [3], a blocking probability vector is used to estimate processor stall time while waiting for a buffer. Our processor queuing model can generate similar blocking probabilities faster than an execution based model, and can quickly re-calculate probabilities as the latencies and contention at the system cause the blocking probabilities to change.

Many system performance issues can be analyzed with statistical synthetic workloads; however, there are some studies, such as buffer replacement algorithms and flow control at memory, that will require memory footprint with timing information. We are looking into augmenting cache simulation traces with timing information derived from the processor queuing model.

For the processor queuing model to provide accurate memory activities for a system model, the inputs to the queuing model must capture the application characteristics at the specific configuration of processors. In particular, cache miss rates and cache-to-cache transfer rates must reflect the larger database

and the greater number of processes executing in a large system.

A processor model is only a submodel in overall system performance analysis. The accuracy of the system submodel is as important to obtain an accurate overall performance estimates. By establishing the accuracy of our processor queueing submodel, we can confidently integrate it into a multiprocessor system model.

## 7. Conclusion

This paper presents a processor queueing simulation model for a modern complex processor. For system design using processors which aggressively exploit instruction and thread-level parallelism, we observe that the processor's memory subsystem, which includes caches and buffers, requires the most detailed modeling for memory intensive OLTP workloads. The model implements easily, executes quickly and produces reasonably accurate results. Another advantage of this type of model is that we can vary the model's workload parameters without the need of actual traces, which are often difficult to acquire for large commercial workloads. It is useful for analyzing future workloads or complex workloads for which traces are not available.

The paper shows that burstiness or clustering of stores in OLTP workload has a significant impact on performance, and should be included when evaluating the buffer sizes. The model is used to analyze the overlap in miss penalty with increasing memory latency. It is shown that the buffer sizes should be tuned by designers to achieve good overlapping when the latency increases. We have discussed a few methods that we are investigating to integrate the processor model with a system model.

## Acknowledgement

We would like to thank Sudarshan Kadambi and Vijay Balakrishnan for providing the cycle accurate simulation results, Robert Lane for reviewing the paper, Robert Cypher for prompting the thought of modeling cache misses overlap, and Anders Landin for insisting to see validation with more stressful workload.

## References

[1]D.H. 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]M. Durbhakula, V. Pai, and S. Adve, "Improving the Speed vs. Accuracy Tradeoff for Simulating Shared-Memory Multiprocessors with ILP Processors," Proc. Intl. Symp. on High Performance Computer Architecture, 1999.

[3]S. S. Mukherjee, et. al., "Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator," Workshop on Performance Analysis and Its Impact on Design, June 1997.

[4]V. Krishnan and J. Torrellas, "A Direct-Execution Framework for Fast and Accurate Simulation of Superscalar Processors," PACT'98, October 1998.

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

[6]V. S. Pai, P. Ranganathan, and S. Adve, "The Impact of Instruction Level Parallelism on Multiprocessor Performance and Simulation Methodology," Proc. Intl. Symp. on High Performance Computer Architecture, 1997.

[7]M. Rosenblum, et. al., "Using the SimOS Machine Simulator to Study Complex Computer Systems," ACM Transactions on Modeling and Computer Simulation, 1997.

[8]E. Schnarr and J. Larus, "Fast Out-of-Order Processor Simulation using Memorization," ASPLOS-8, October 1998.

[9] E.Schnarr, M. Hill and J. Larus, "Facile: A Language and Compiler for High-Performance processor Simulators", PLDI'01, 2001.

[10]D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood, "Analytical Evaluation of Shared-Memory Systems with ILP Processors," Proc. Intl. Symp. on Computer Architecture, 1998.

[11]Sun Internal Presentation, R. Cypher.

[12]Sun Internal Presentation, N. Kosche and Q. Jacobson.

[13]TPC Benchmark C Standard Specification, Transaction Processing Performance Council.

[14]D. Willick and D. Eager, "An Analytical Model of Multistage Interconnection networks," in Proc. ACM Sigmetrics, May 1990.

[15]M. Vernon, E. Lazowska, and J. Zahorjan, "An Accurate and Efficient performance Analysis Technique for Multiprocessor Snooping Cache Consistency protocols," International Symposium on Computer Architecture, 1988.