CMP Memory Modeling: How Much Does Accuracy Matter?

Sadagopan Srinivasan Li Zhao Brinda Ganesh Bruce Jacob* Mike Espig Ravi Iyer
Systems Technology Lab, Intel Corporation
* University of Maryland, College Park
Contact: Sadagopan.srinivasan@intel.com

Abstract: As Chip-multiprocessor (CMP) become the ubiquitous architecture, especially for commercial servers targeting throughput-oriented applications, processor manufacturers are likely to integrate increasing number of cores on-die. Designing and developing these CMP architectures involves studying a number of options for on-die interconnect, cache and memory system while optimizing for both power and performance. Simulation-based study is widely adopted for the design space exploration for these systems. Although most existing CMP simulators have detailed cache and interconnect models, they use simplistic memory models that use either a fixed latency to the memory sub-system or a simple queuing model which adds bandwidth constraints to the fixed latency approach. In this paper, we demonstrate the necessity for a cycle-accurate memory model for CMP architecture. We study three types of memory models: (1) Fixed latency model, (2) simple queuing model and (3) detailed cycle-accurate model. We show that the performance difference among them is increased as the number of cores is increased on-chip. We also find that optimization studies done using simplistic models can lead to erroneous conclusions. Our studies show that the performance difference between simplistic models and accurate memory controller can be as high as 65% for memory optimization studies.

1. Introduction

The scaling limitations of uni-processor and availability of large silicon area due to reduced transistor size has lead to increased number of cores on a chip. The cost of extracting more instruction level parallelism (ILP) from a single thread/core is becoming expensive due to complex logic, wider issue width and even more accurate branch predictors. These factors have fueled the growth of chip multi-processors (CMPs), also known as multi-core processor. These CMPs are becoming the ubiquitous architecture for commercial servers targeting throughput-oriented applications [1].

The emergence of CMPs has lead to increased exploitation of the thread-level parallelism. Furthermore, independent processes in a system can be executed in tandem on different cores for faster response time, and to improve the overall throughput. The simultaneous execution of multiple processes/threads increases the memory bandwidth demand, i.e. the increased number of cores aggravates the memory wall problem.

The other factor contributing further to the memory bandwidth problem is the slowdown in growth of number of pins per die and pin bandwidth. The inadequate growth in memory bandwidth will further aggravate this problem as we move to increased integration.

To explore CMP design space and propose optimization techniques, simulation methodology is widely adopted. Most modern CMP simulators though have a detailed cache and interconnect models, use a simplistic memory model [2][3][4]. The memory system is assumed to be a fixed latency model [2] or a simple queuing model. In the fixed latency model, all memory requests experience the same latency irrespective of bandwidth constraints. For bandwidth-constrained systems such as multi-threaded processors, the fixed latency model would give overly optimistic results. A slightly improved model is a queuing model which has bandwidth constraints, with a specific arrival and service rate for memory requests. The queuing model is based on M/M/1 model where the arrival and service rate are assumed to be Poisson distribution.

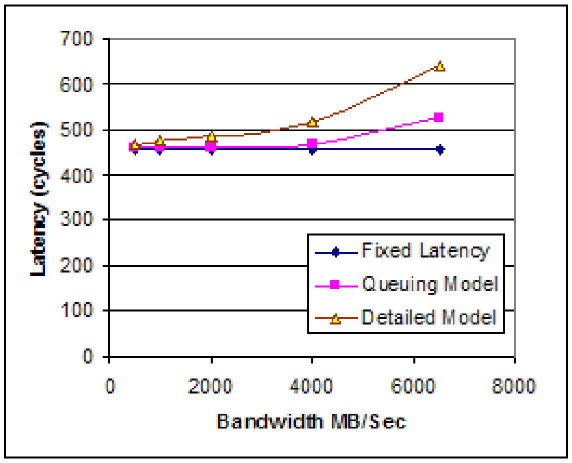


Figure 1. Latency response for various memory models

Figure 1 illustrates the memory response time at different throughput requirements for the three memory models. Compared to a detailed memory model, the two simplistic models behave similar to it at lower bandwidth requirement, but do not faithfully track it at higher bandwidth. Though the queuing model fares slightly better than the fixed model in tracking the detailed model's behavior, it still underestimates the memory latency by as much as 25% for the maximum throughput. This is due to the queuing model's inability to capture the memory contention overhead, which we explain later, increases significantly with the throughput. The simplistic models do not work as well for memory intensive workloads as they do for compute intensive workloads.

In this paper, we study these three memory models in detail. Applications that are memory-bound can show artificial improvement in performance when using simplistic models, but will not result in true performance gain in an actual system which will have a cycle-accurate model. We show that the performance difference between the two models can be as high as 15% and can increase up to 65% for memory optimization studies, such as prefetching. This behavior can lead to incorrect conclusions about certain optimization techniques and result in substandard products.

We also show that irrespective of memory optimization techniques, using simplistic models can result in incorrect performance projections for multi-core systems. We observed that the difference in IPC between simple latency model and cycle-accurate model (with rest of the system being same for both models) is 2% for a single core, and increases to 15% for 8 cores. This can lead to incorrect conclusions about relative performance gains as the number of cores is increased.

The rest of the paper is organized as follows. Section 2 describes our motivation. Section 3 describes our simulation methodology for the three memory models. Section 4 describes the results, and section 5 describes the related work. We conclude in section 6 with our findings.

2. Motivation

In this section, we explain the latency response of a detailed cycle accurate model similar to [5] and highlight the importance of memory subsystem for CMPs. Figure 2 illustrates the memory sub-system response at different throughput requirements measured for dual channel DDR3-800 with closed paging policy. The maximum sustained bandwidth for this system is around 7GB/s. *Maximum sustained bandwidth* is the maximum bandwidth observed in the simulation and is different from the theoretical maximum. In our study, this has been observed to be around 70-80% of the theoretical maximum for server workloads and depends on various factors such as read-write ratio, paging policies, address mapping etc. The bandwidth-latency curve consists of three distinct regions.

Constant region: The latency response is fairly constant for the first 40% of the sustained bandwidth. In this region the average memory latency almost equals the idle latency in the system. Idle latency is the default cost of the memory operation i.e. the cost of opening the page, reading data out of the open page, and returning it to the processor. The system performance is not limited by the memory bandwidth in this zone, either due to applications being non-memory bound or due to excess bandwidth availability.

Linear region: In this region, the latency response increases almost linearly with the bandwidth demand of the system. This region lies for throughputs in the range of 40% to 80% of the sustained maximum throughput. The average memory latency starts to increase due to contention overhead introduced in the system by numerous memory requests. The performance degradation of the system starts in this zone, and the system is claimed to be fairly memory bound.

Exponential region: This is the last region of the bandwidth-latency curve. This region exists between 80%-100% of the sustained maximum. In this zone the memory latency is dominated by the contention latency which can be as much as twice the idle latency or more. Applications operating in this region are completely memory bound and their performance is limited by the available memory bandwidth.

The figure clearly illustrates the need for a system to operate in the constant region or at least the linear region. However, due to the increased bandwidth demands of multi-core systems and the hurdles faced in scaling memory bandwidth, systems will be forced to operate in the linear and exponential region more frequently.

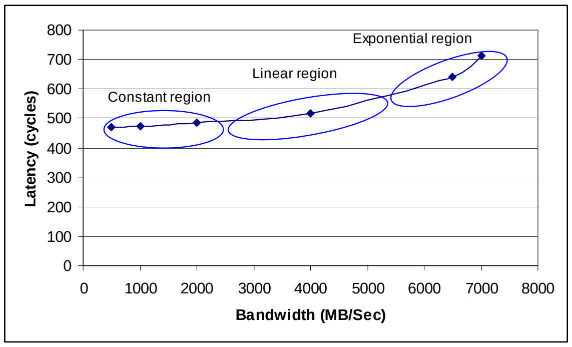


Figure 2. Memory Bandwidth Vs Latency curve

The memory bandwidth problem will lead to significant reduction in performance gain as the number of threads is increased. Figure 2 illustrates this non-linear performance scaling for SPECJbb [6], a server workload. We simulated various threads with each having a private 16KB L1 cache (separate instruction and data cache), each set of 8 threads shared the 512KB L2 cache and all threads shared the last level cache. The last level cache size (L3) was increased proportionally from 2MB to 32MB for 8 to 128 threads. The maximum available memory bandwidth was set to 52 GB/Sec.

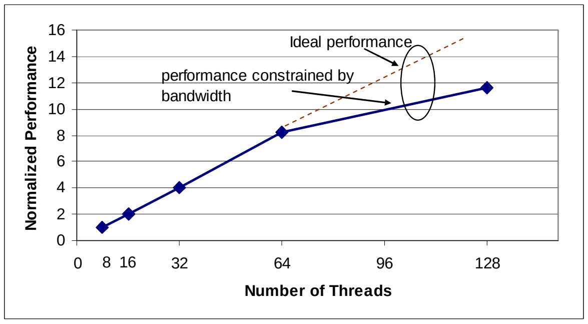


Figure 3. Performance scaling over threads for SpecJbb

The system performance scales linearly from 8 threads to 32 threads. Beyond 32 threads, this performance gain tapers down because of the increased average memory latency of the system. The memory latency increases exponentially, as explained above, for a large number of threads (greater than or equal to 64 in this case). This contributes to the non-linear increase of system performance with the number of threads. This non-linear scaling in performance can be captured only in a system which faithfully models all memory system interactions.

3. Methodology

In this section, we explain our simulation methodology, memory models and simulation parameters in detail.

3.1 Simulation Framework

We use a trace-driven platform simulator called ManySim [3] to evaluate CMP platforms. ManySim simulates the platform resources with an abstracted core. ManySim contains a detailed cache hierarchy model, a detailed coherence protocol implementation, an on-die interconnect model and a queuing memory model as described in previous section.

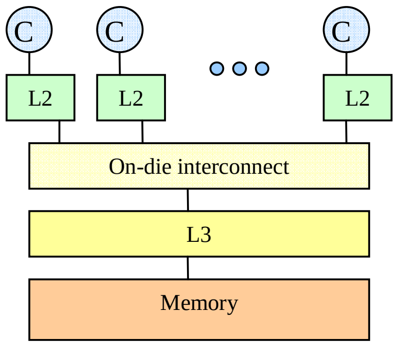


Figure 4. Multi-core Architecture

Figure 4 shows our multi-core architecture model. Each core in our model has a private L1, a private L2 and all cores share a distributed L3 cache as shown in figure 4. To study the impact of different memory models on estimating memory performance we enhanced the ManySim simulator to run with three different memory models including a detailed memory model in this study. We refer the reader to [3] for a more detailed description about the methodology.

(1) Fixed Memory Model: In this model, all memory requests incur the same delay irrespective of the requested system bandwidth, access pattern, ratio of read to write requests etc. There is no concept of bandwidth limitation in this model. This does not address the memory contention overhead either. Memory contention overhead is the effect experienced by a memory request when the memory controller's transaction queue is full. In this scenario, a memory request has to contend with other requests to get serviced and becomes more pronounced at higher bandwidths of the system. This is the most simplest of all models, and hence the fastest. This is used in most uni-processor CPU simulators such as Simplescalar [7], alpha-sim [8] and multi-processor simulator like GEMS [2].

(2) Queuing Memory Model: This model is based on the memory requests arrival and servicing rate. The arrival and service rate are assumed to be Poisson distribution in our study. Unlike the fixed latency model, the queuing model is able to capture the effects of bandwidth-constraints on memory latency albeit to a limited extent as shown in later sections. This model still does not capture the effects introduced in the system due to contention among memory requests accurately. This model is faster than the accurate memory controller but slower than the idle latency model.

3) Detailed Memory Model: The memory controller is a detailed cycle-accurate model that supports DDR and FBD protocols similar to DRAMSim [5]. The model supports various scheduling algorithms such as read first, write first, adaptive etc. The scheduling algorithm used in this study is an adaptive scheduling algorithm. This policy gives priority to read requests over write requests as long as the number of outstanding writes is below a threshold. The threshold is set to be 2/3rd of the write queue size. The model also provides the flexibility to vary the address mapping policies, number of ranks, DIMMs etc. in the system. In our studies we call this *Accurate Cycle Latency Model* (ACLM). The detailed memory controller is the most cycle accurate of all models and captures the memory system behavior completely. This is also the slowest among all models.

Both the fixed and queuing models need to specify service latency for a request. This latency value is used by the *Fixed Memory Model* as the latency value for all transactions, while it is used in the *Queuing Memory Model* as the basic memory latency without any bandwidth impact. This provides two additional variations:

- *Idle Latency Model* (ILM), where the minimum round trip time for a memory request is equal to the idle latency of the accurate memory controller.
- *Average Latency Model* (ALM), where the minimum round trip time for any memory request is equal to the average memory latency of the accurate memory controller i.e. the idle latency of this model is equal to the average latency of the accurate model. Average latency for each workload is computed for the entire simulation period using an accurate memory controller.

Based on the memory model and latency value it uses, we have the following four types of simplistic memory models:

- Simple Idle Latency Model (SILM): This fixed latency model uses the idle latency value as the memory latency.
- Simple Average Latency Model (SALM): This fixed latency model uses the average latency of the cycle accurate model for the memory latency.
- Queue Idle Latency Model (QILM): In this model the minimum latency for a memory request is equal to the idle latency of the AMC.
- Queue Average Latency Model (QALM): This is the type of queuing model where the minimum latency of a memory request is equal to the average latency of the cycle accurate model.

Table 1 summarizes the behavior of these various memory models.

Memory Models	**Bandwidth Limitations**	**Memory Contention overhead**	**Simulation speed**
Fixed Memory model	N	N	Fastest
Queuing Memory model	Y	N	Medium
Detailed Memory Model	Y	Y	Slow

Table 1. Memory Model Comparison

Table 2 summarizes the various simulation parameters in our study. We varied the number of threads (threads are synonymous to cores in our studies) in the platform from 1-16. The L2 cache slice/core is 256KB and scaled linearly with the cores. The L3 cache size was 1MB/core and was scaled linearly as well. We set the simplistic model bandwidth to be the maximum sustained bandwidth of the detailed DRAM model. We also modeled different memory channels (1, 2 and 4) for the detailed model and had corresponding service latencies for the simplistic models.

Parameter	**Variations**
Number of cores/threads	1, 2, 4, 8, 16
Shared L2 cache size	256KB - 2MB (scaled linearly with cores). 8-way, 64-byte line size
Shared L3 cache size	1MB - 8MB (scaled linearly with cores). 16-way 64-byte line size
Simplistic Memory Model	Fixed and Queuing Model
Detailed DRAM Model	DDR3 800 with support for 1, 2 and 4 channels, read and write queue size of 42, adaptive scheduling

Table 2. Simulation Parameters

3.2 Workloads

We used server benchmarks in our study as these are some of the important classes of applications to exploit the performance benefits of chip multiprocessors. The memory traces were captured from a four socket dual core Pentium 4 machine. The traces were captured from significant points in the workload that reflect the benchmarks behavior accurately.

On-line transaction processing (OLTP) is represented using TPC-C [9]. TPC-C simulates a complete computing environment where a population of users executes transactions against a database.

ERP is represented using sales and distribution benchmark, the SAP SD 2-tier benchmark [10]. Transactions in this application involve creating orders, creating deliveries for orders, displaying orders, changing options, listing and creating invoices.

SPECjbb2005 [6] is a Java-based server benchmark that models a warehouse company that serve a number of districts (much like TPC-C). This workload is intended to test the performance of JVM components including garbage collection and runtime optimization.

SPECjAppServer2004 [11] is a multi-tier benchmark for measuring the performance of Java 2 Enterprise Edition (J2EE) technology-based application servers. It is an end-to-end application which exercises all major J2EE technologies implemented by compliant application servers.

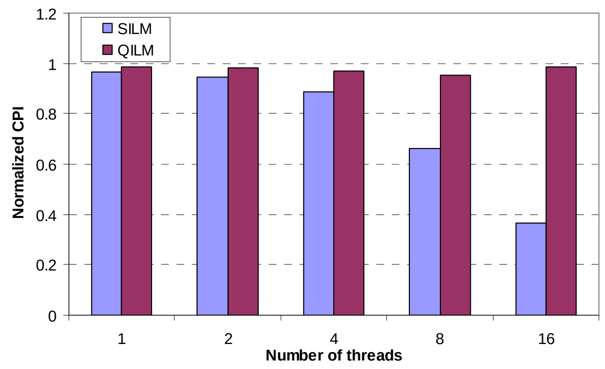
4. Performance Evaluation

In this section, we present the performance evaluation using the different memory models. We present the impact on CPI, throughput projection and also show the impact of various models on memory optimization techniques like prefetching. Our results show that all benchmarks exhibit similar trend, hence we present the results for SPECJbb in this section.

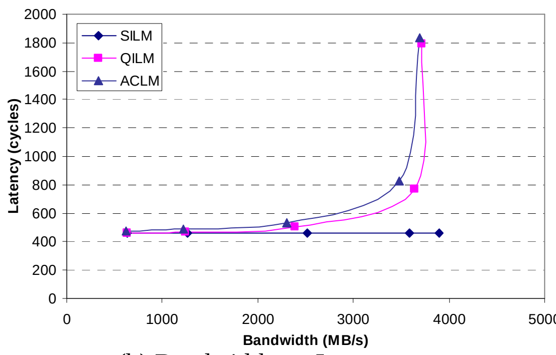
4.1 Impact on CPI

Figures 5, 6 and 7 illustrate the performance of SILM and QILM for different number of threads with 1, 2 and 4 memory channels respectively. The y-axis shows the CPI normalized to the detailed memory model. Figure 6(a) shows that for single threaded workloads the CPI obtained from both models are nearly identical to that obtained from the detailed model. However, increasing the number of threads, results in an increase in the difference between the CPI values predicted by the detailed model and those obtained from the SILM. The CPI is off by 35% and 62% respectively for 8 and 16 threads. This large error in estimation is because SILM makes an unrealistic assumption of infinite bandwidth. SILM behaves close to the cycle accurate model only in the *constant region* of the bandwidth latency curve as shown in figure 5(b).

QILM behaves different from SILM, and the difference with detailed model is minimal. The error increases to 5% when the number of threads is 8, but reduced to 2% for 16 threads. This is because of the bandwidth constraint imposed by the queuing model is identical to the detailed model in the operating region of the applications. QILM model behaves similar to the detailed model at the two extreme end of the bandwidth requirement (i.e. *constant and exponential region*) as show in latency response graphs and diverges from it in the linear region. Hence the performance difference with detailed model first increases and then decreases.



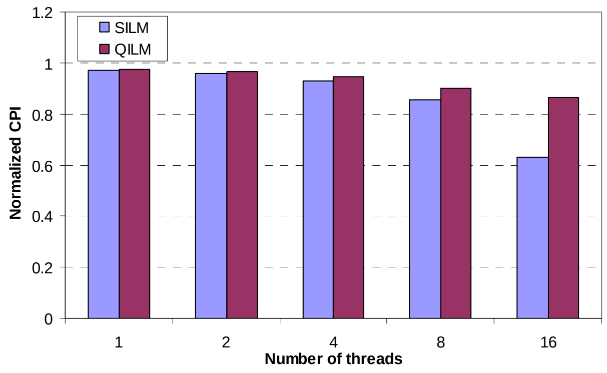
(a) CPI Normalized to a detailed model



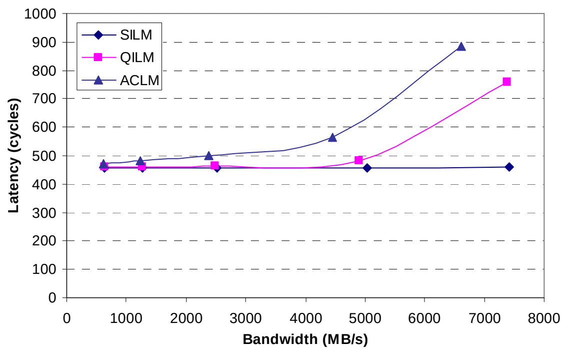
(b) Bandwidth vs. Latency response

Figure 5: **Single Channel performance results**

As shown in figure 6, the dual channel configuration gives similar trend for both SILM and QILM. Both models have increased difference from the detailed model when the number of threads is increased. However QILM still performs better. The difference is 15% and 57% for QILM and SILM respectively. Since the 2 channel configuration provides more bandwidth than 1 channel, the applications operate in the linear region of the bandwidth-latency curve.

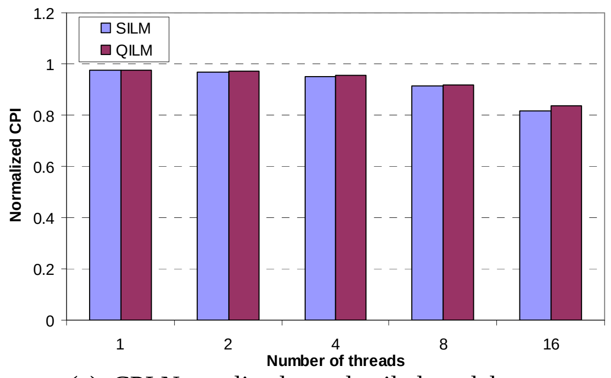


(a) CPI Normalized to a detailed model

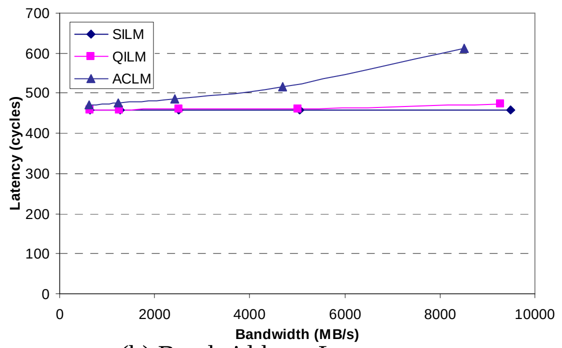


(b) Bandwidth vs. Latency response

Figure 6: **Dual Channel performance results**



(a). CPI Normalized to a detailed model



(b) Bandwidth vs. Latency response

Figure 7: **Quad Channel performance results**

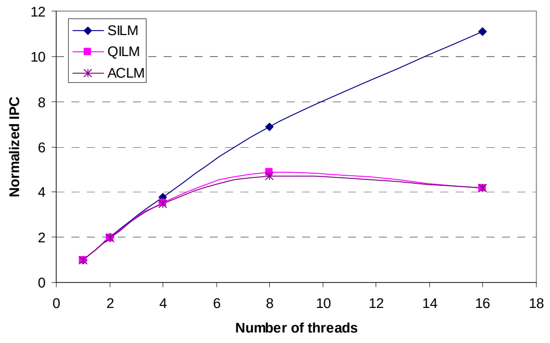
The difference between QILM and ACLM is maximal in this zone, and hence the performance difference is more for QILM with 2 channels compared to 1 channel. It is due to the same reason, increased bandwidth availability, SILM fares better with 2 channel configuration than 1 channel.

We observe a similar trend continuing with 4 memory channels as shown in figure 8. The difference from the detailed model is increased to about 19% when the total number of threads is 16 for both the models. Since 4 channels provide even more bandwidth as opposed to 2 channels, the difference between the two models and detailed model becomes comparable.

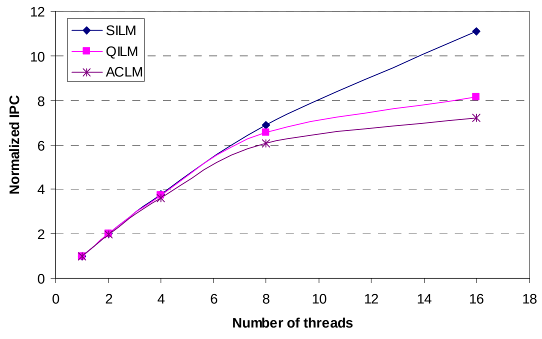
4.2 Impact on Throughput Projection

We also looked at the throughput projection as we increased the number of threads with the three memory models. Figure 8 (a), (b) and (c) shows this data for 1, 2 and 4 memory channels respectively. The y-axis shows the performance improvement normalized to a single thread. We can see that with single memory channel, QILM has the same curve as the detailed model: the memory throughput is increased to its maximum with 8 threads, but starts to reduce as we keep increasing the number of threads due to the increased memory traffic. QILM is able to capture the bandwidth constraints of a real machine accurately in a tightly constrained system. SILM, on the other hand has a very different behavior. The throughput keeps increasing as the number of threads is increased. This is due to the fact lack of bandwidth constraint in the model. This model works fine in a memory unconstrained environment but not so well when memory is overloaded. Hence SILM can capture the performance improvement from 1 to 4 threads as well as ACLM but not so for higher threads.

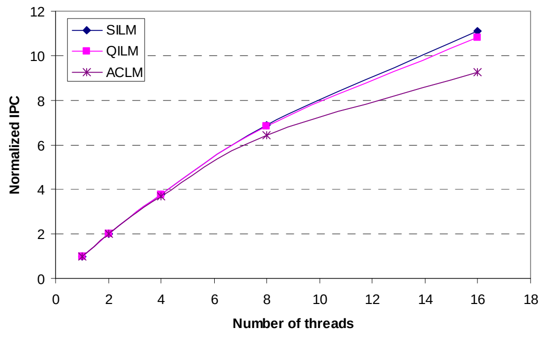
QILM matches the detailed model with 1 memory channel, but it shows different behavior as we increase the memory channels, as shown in figure 8(b) and 8(c). With 4 memory channels, QILM behaves more like the SILM. This is because of the QILM diverging from the detailed model in the linear region of the bandwidth-latency curve as shown in Figure 8(c).



(a) Single Memory Channel



(b) Dual Memory Channel



(c) Quad Memory Channel

Figure 8: **Throughput projection for various memory channels with different threads**

4.3 Average Latency Results

In the above results both the SILM and QILM used the ACLM idle latency as the memory service delay in their models. It is expected that using the average latency can provide more accurate results. Therefore, we ran the benchmarks with ACLM, computed the average latency, and used them in these two models.

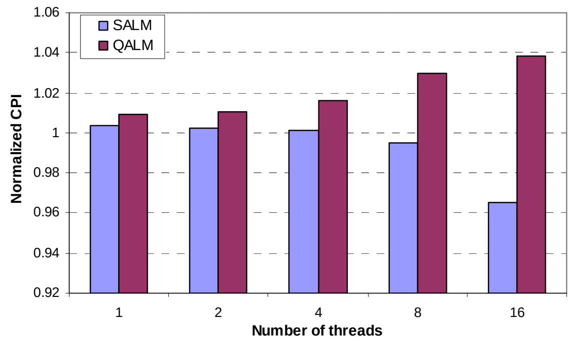


Figure 9. **CPI normalized to detailed model for SALM and QALM with dual memory channels**

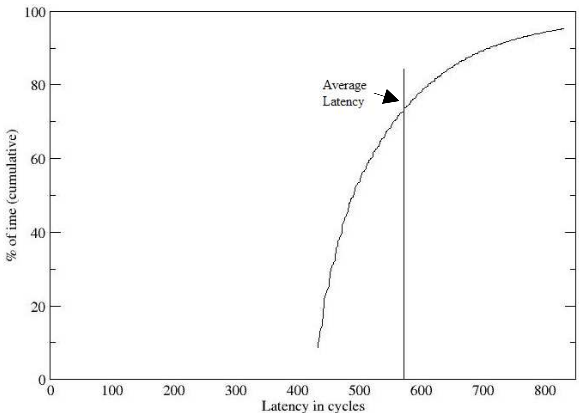


Figure 10. **Memory latency distribution**

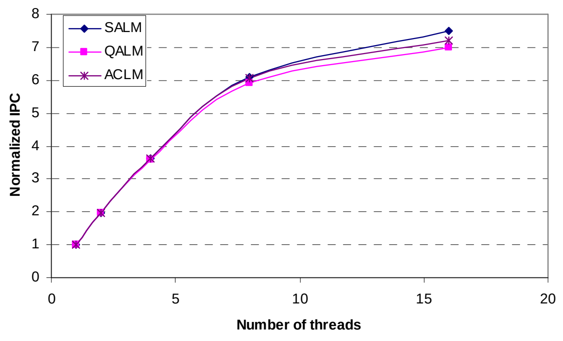


Figure 11. **Throughput projection for Dual memory channels with different threads**

Figure 9 shows the CPI of SALM and QALM memory models normalized to ACLM. Here, the SALM model performs better than the idle latency model and varies from over predicting the CPI by 0.5% for 1 thread to under predicting it by 5% for 16 threads. QALM over predicts the CPI for all case with the maximum difference being 4% for 16 threads. Over prediction of performance happens due to the fact that most memory requests in a real system don't necessarily experience the average memory latency as shown in figure 10. This figure shows that almost 65% of the requests experienced less than the average latency. Since we assume the same memory latency for all requests, the average latency models tend to be more conservative. Hence, the average latency models lead to over prediction of CPI.

Figure 11 shows the throughput projection using the two average latency models. The y-axis shows the performance improvement normalized to a single thread. Both these models track the detailed ACLM model well. Even the SALM fixed latency model, with no bandwidth constraint, captures the trend accurately due to a more accurate average latency being used in the model. This is due to the fact that we capture the average latency for each thread configuration using the ACLM and plug it back into the service latency of the simplistic models. As we show in the next section, computing the average latency from ACLM for each thread/benchmark configuration is cumbersome, but the model behaves very close to a detailed model.

4.3.1 Challenges in Average Latency Model

Though the average latency models capture the memory system behavior well, we highlight the challenges in computing the average latency in this section.

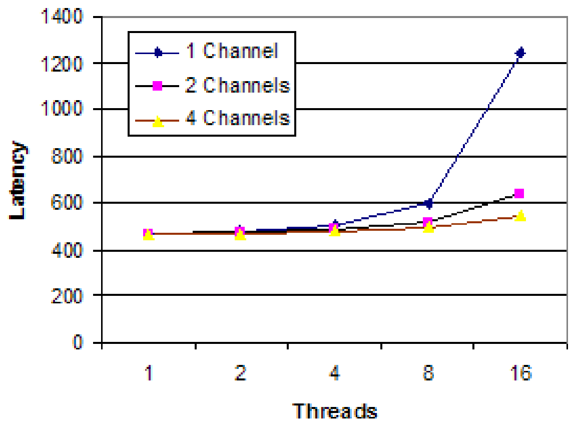


Figure 12. **Average memory latency for various memory channels**

Figure 12 shows the average memory latency for the three memory channels for different threads. This graph illustrates that the average latency can vary from about 450 cycles to 1250 cycles, a factor of 2.6X, for different number of threads and memory channels. The average latency is subject to change for each benchmark as well. This is because the average memory latency in a system depends on various factors such as number of DIMMs, number of banks, read-write ratio of the memory requests, scheduling algorithms, number of cache misses etc. This is a big challenge as the number of such parameters is huge and variation in the latency can vary a lot as well. A solution would be to use Monte Carlo approach of running various benchmarks with different memory configurations and using the average latency thus computed in simplistic models. This still cannot guarantee an accurate performance prediction as any change in the platform architecture can affect the average memory latency. Hence, for our base case we used the idle latency of the system over average latency.

4.4 Impact of memory optimization techniques

In this subsection we study the impact of simplistic models on memory optimization studies:prefetching. Prefetching has been well established to reduce memory latency in the system [12]. We studied the performance of stream prefetchers in the last level cache with a stream depth of 5.

Figure 13 illustrates the IPC difference of all the simplistic memory models (SILM, SALM, QILM and QALM) with respect to ACLM for dual memory channel. We observe that the idle latency models (SILM) under predict the performance by up to 65% (and 13% for QILM) for 8 threads, a trend shown earlier. This is due to the lack of bandwidth constraint in SILM, and inability to capture contention overhead in QILM.

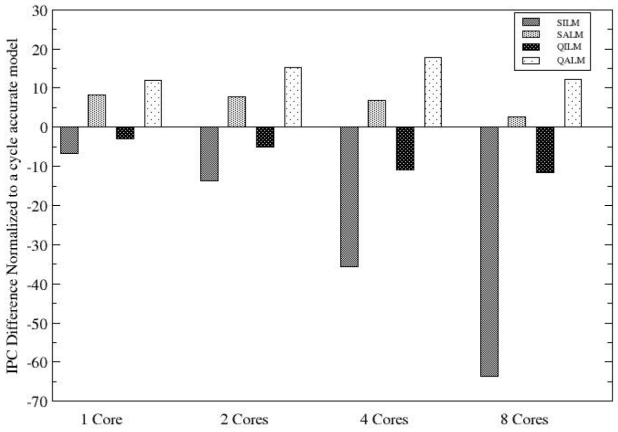


Figure 13. **IPC difference of simplistic models normalized to ACLM for SpecJbb**

The average latency models (QALM) over predict the performance by up to 18% (and 8% for SALM), as shown earlier, due to most requests experiencing a latency less than the average latency in a real system. These results highlight their inability to capture actual platform behavior.

Memory Model	1 thread	2 threads	4 threads	8 threads
ACLM	**4.0%**	**-3.7%**	-28.4%	-56.9%
SILM	10.5%	**8.8%**	5.7%	5.3%
SALM	**-3.8%**	-10.7%	-32.9%	-57.7%
QILM	6.19%	**0.3%**	-22.0%	-54.1%
QALM	**-6.7%**	-15.6%	-38.1%	-61.7%

Table 3. Performance Improvement over threads with prefetching

Most studies focus on relative performance improvement rather than absolute numbers. Table 3 shows the performance improvement for various threads using different memory models with prefetching (i.e. each model is compared against itself without prefetching and the resulting performance improvement is shown). We notice that the performance trend for SILM is vastly different from ACLM. SILM shows performance benefits of 5% with prefetching for 8 threads, whereas ACLM shows a significant degradation in performance for the same case. As explained earlier, this is due to the lack of bandwidth constraint in SILM. Similarly, QILM shows performance benefits for 2 threads whereas ACLM shows benefits only for single thread scenario and degradation for others due to the system operating in the exponential region of the bandwidth latency curve. The average latency models, due to their conservative nature, project performance degradation even for the single threaded case. These results show that one might reach erroneous conclusions about relative performance improvement using simplistic models.

5. Related Work

There are various studies that highlight the need for accurate architectural models to evaluate the system performance. Alameldeen and Wood identified the performance variability as a major challenge for architectural simulation studies for multi-threaded workloads [13]. Variability in this study refers to the differences between multiple estimates of a workload performance. The impact of variability on multi-threaded workloads can be extended to chip-multiprocessors. Alameldeen et al. also have characterized commercial workloads dependency on non-determinism [14]. The authors propose a methodology that uses pseudo-random perturbations and standard statistical techniques to compensate for the non-deterministic effects.

Desikan et al. highlight the experimental error that arises from the use of non-validated simulators in computer architecture research in a uni-processor environment [7]. This work describes ways to reduce the error by considering specific aspects of the pipeline. A similar study involving multiprocessors was studied in [15] by Gibson et al. The authors have compared their simulator with an actual hardware for FLASH based systems. This paper studies the source and magnitude of error in a range of architectural simulators by comparing the simulated execution time of several applications to their execution time on the actual hardware being modeled.

Krishnan and Torellas examined experimental errors in multiprocessor simulations due to simple processor models [16]. They propose a novel direct-execution framework that allows accurate simulation of wide-issue superscalar processors without the need for code interpretation.

Cain et al. discusses the issues of precision and accuracy in simulation [17]. Their work highlights the operating system effects on both commercial and SPECint workloads. They also show that incorrect speculative path in a simulation environment is unimportant for these benchmarks and show the I/O effects on simulation accuracy for uni-processors.

Simulation errors by selecting particular program phases were investigated by Sherwood et al. [18]. This study proposes a solution to address this problem by selecting basic block distribution that represents the entire program’s execution across different architectural metrics such as branch miss rate, IPC, cache miss rate etc. This approach is based upon using program’s profile code structure to uniquely identify the different phases of execution in the program.

Oskin et al. introduce a hybrid processor simulator that uses statistical models and symbolic execution to evaluate design alternatives [19]. This simulation methodology allows for quick and accurate contour maps to be generated to the performance space spanned by design parameters.

Most of these aforementioned studies focus on the core and omit the memory system. Our work highlights the need for an accurate modeling of memory system in CMPs where the memory plays a crucial role in determining the performance.

6. Conclusion

In this work, we presented various simplistic memory models and highlighted the drawbacks of using them. One of the main arguments in favor of such models has been that they are sufficient to compute the performance difference between various configurations, though may not be useful for absolute values. Our studies show that these models can be wrong both in absolute performance numbers and relative performance comparison between different configurations.

Our studies show scenarios wherein the simplistic models either over-predict or under-predict the system performance with respect to cycle accurate model. We also show that using simplistic models can lead to wrongful conclusions in terms of performance projection as shown with SALM and QALM. These simplistic models predicted performance degradation with prefetching for single thread system whereas the ACLM predicted improvement. Under-predicting the performance can lead to over designing the system, and will render it expensive. Over-predicting the performance can lead to system being ineffective due to not being able to meet the performance constraints of applications in a real world environment. Both these cases are causes of concern due to simplistic models.

Our results show that as the system complexity grows more accurate models are needed to evaluate the system performance. The ease of use and speed offered by the simplistic models are easily offset by the inaccurate results produced by them and may not serve their purpose. An optimized solution would be to have a hybrid memory model wherein the detailed model co-exists with the simplistic models and based on the platform throughput requirements the appropriate memory model is chosen. This kind of modeling can be both fast and accurate.

References

1. John D. Davis, James Laudon, Kunle Olukotun, “Maximizing CMP Throughput with Mediocre Cores”, *PACT* 2005
2. Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, “Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset”, *Computer Architecture News (CAN)*, September 2005
3. L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell, “Exploring Large-Scale CMP Architectures using Manysim”, *IEEE Micro*, vol. 27, issue 4, pp. 21-33, August 2004
4. P. G. Emma, A. Hartstein, T. R. Puzak, and V. Srinivasan, “Exploring the limits of prefetching”, *IBM Journal of Research and Development*, vol. 49, issue 1, pp. 127-144, January 2005
5. David Wang, Brinda Ganesh, Nuengwong

Tuaycharoen, Katie Baynes, Aamer Jaleel, and Bruce Jacob, "DRAMsim: A memory-system simulator", *SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 100-107. September 2005

6. SPECjbb2005 Java Business Benchmark, available online at http://www.spec.org/jbb2005/
7. R. Desikan, D. Burger, and S. W. Keckler, "Measuring experimental error in microprocessor simulation", in 28th *ISCA* 2001
8. Doug Burger and Todd M. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Department of Computer Sciences, University of Wisconsin-Madison, June 1997
9. "*TPC-C Design Document*", www.tpc.org/tpcc/
10. Sap America Inc., "*SAP Standard Benchmarks*", *http://www.sap.com/solutions/benchmark/index.epx*
11. http://spec.org/jAppServer2004/
12. N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", *In Proceedings of the 17th International Symposium on Computer Architecture*, pp. 364-373, May 1990
13. A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads", in *9th HPCA*, 2003
14. A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads", in *9th HPCA*, 2003
15. J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich, "Flash Vs. (Simulated) Flash: closing the simulation loop", in *ASPLOS* 2000
16. V. Krishnan and J. Torellas, "A Direct execution framework for fast and accurate simulation of superscalar processors", in *proceedings of International Parallel Architecture and Compilation Techniques*, 1998
17. H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H. Lipasti, "Precise and accurate processor simulation", in proceedings of *fifth workshop on computer architecture evaluation using commercial workloads*, pp. 13-22, 2002
18. T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications", in *proceedings of International Parallel Architecture and Compilation Techniques*, 2001
19. M. Oskin, F. T. Chong, and M. Farrens, "HLS: Combining statistical and symbolic simulation to guide microprocessor designs", in 27th *ISCA*, 2000