ISR TECHNICAL REPORT 2012-07

Producing Reliable Full-System Simulation Results: A Case Study of CMP with Very Large Caches

Mu-Tien Chang
Ishwar Bhati
Jim Stevens
Paul Tschirhart
Peter Enns
Daniel Gerzhoy
James Greensky
Shih-Lien Lu
Bruce Jacob

Producing Reliable Full-System Simulation Results: A Case Study of CMP with Very Large Caches

Mu-Tien Chang[*], Ishwar Bhati[*], Jim Stevens[*], Paul Tschirhart[*], Peter Enns[*], Daniel Gerzhoy[*]
Zeshan Chishti[†], James Greensky[†], Shih-Lien Lu[†], Bruce Jacob[*]
[*]University of Maryland, College Park
[†]Intel Corporation

***Abstract*—The greater detail and improved realism of full-system architecture simulation makes it a valuable computer architecture design tool. However, its unique characteristics introduce new sources of simulation variability which could make the results of such simulations less reliable. Meanwhile, the demand for more levels of cache and larger caches has increased to improve the system power and performance. This paper presents techniques to produce reliable results in full-system simulation of CMP computer systems with large caches. Specifically, we propose the detailed emulation replay warmup technique to deal with cold or incompletely warmed up large caches. We also propose the region of interest synchronization technique to prevent simulating non-representative phase when running multi-program workloads. Furthermore, we quantify the variation reduction one can achieve when using processor affinity and checkpointing. Finally, we show that by applying all four of these simulation techniques, the simulation variability is limited to less than 1% and the simulation results are therefore more reliable.**

I. Introduction

Performance evaluation is fundamental to computer architecture design. Due to the complexity of modern computer systems, architectural simulation has become the major performance evaluation method since it is more accurate than analytical models and at the same time is not as costly as hardware development. In particular, full-system simulation is capable of simulating an entire computer system, including user-level application, operating system (OS), and hardware, thus providing a more realistic framework.

Nevertheless, full-system simulation has limitations. One problem is that since it simulates computer systems in detail, it is slower than simpler models of simulations such as trace-driven simulations [24] or Pin-based approaches [7]. The consequence of trading off simulation speed for accuracy is that users can only run truncated workloads (fewer numbers of instructions) in order for a full-system simulation to complete in a reasonable time. As a side effect of simulating fewer instructions, full-system simulations can produce non-representative results, which can be caused by different OS scheduling decisions or non-coherent initial simulation states [5]. This can lead to incorrect conclusions based on results that are either inaccurate or exhibit large variations.

On the other hand, computer systems with more levels of cache and larger cache capacities have emerged for better performance and energy-efficiency. For instance, the IBM Power7 [17] has a 32 MB L3 cache; Yun et. al [27] presented a 3D architecture with a 128 MB 3D-stacked L3 eDRAM-based cache; Quereshi et. al [23] proposed a hybrid DRAM/PCM memory system that uses a 1 GB DRAM cache for the high-density PCM main memory. In this paper, we present techniques to produce reliable full-system simulation results in the context of CMP computer systems with large caches, using a quad-core architecture with a 512 MB DRAM cache for the hybrid main memory system as the case study. We make the following main contributions:

- We identify four sources of simulation inaccuracy or non-determinism: First, *the cold cache problem* occurs when a simulation begins from a representative phase but ignores the start-up period of the program. Traditionally, a cache warm-up period is used as the solution to this problem, but very large caches make warm-up inefficient and, in some cases, ineffective. Second, *the non-representative phase problem* in multi-program workloads happens if only a subset of the programs switches to detailed simulation at the region of interest (RoI), while other programs switch to detailed simulation at random times. This results in less reproducible experiments because each program in the workload is not starting from a known point and therefore each experiment is different. Third, *the workload imbalance problem* happens as a side effect of full-system simulation with an OS scheduler. Since the OS scheduler state is unknown when full-system simulation begins and the scheduler makes decisions every few milliseconds [28], task switching can have a significant effect on the reproducibility of short period simulations. Finally, *non-deterministic simulation starting state* leads to increased simulation variability. For example, under normal operation the OS will introduce variability in the memory locations of data. Therefore, each experiment will start from a different system state resulting in unreproducible experiments.
- We propose *the detailed emulation replay warmup technique* to deal with cold or incompletely warmed up large caches. This technique works by capturing a trace of the cache accesses that occur from the boot of the simulated system to the RoI in fast emulation mode and then replaying them through the cache sub-simulation only to

quickly warm up the entire cache.

- We propose *region of interest synchronization* by using interprocess communication to create a barrier, which guarantees each program of a multi-program workload switches to detailed simulation mode at its RoI.
- We quantify the simulation variation reduction when using *processor affinity* and *checkpointing*. Processor affinity forces tasks to be balanced among processors. Checkpointing ensures each simulation has the same simulation starting state.
- We show that by applying these techniques to our case study, a multi-core system with a large DRAM cache, the variability of the full-system simulation is reduced to less than 1%.

The remainder of this paper is organized as follows. Section 2 describes the non-deterministic simulation problems. Section 3 presents our full-system simulation framework and multi-program workloads. In section 4, we elaborate upon our simulation techniques and demonstrate their effects. Section 5 describes related work. Finally, we conclude this paper in section 6.

II. Simulation Non-Determinism

In this section, we describe four sources of simulation inaccuracy or non-determinism when simulating a full-system multi-core architecture with large caches. They are:

1) The large cold memory problem, which happens when ignoring warmup or using traditional warmup methods. Ignoring warmup results in simulation inaccuracy. On the other hand, traditional warmup techniques are inefficient because they require long simulation periods.
2) The non-representative phase problem in multi-program workloads happens when any of the programs switch to detailed simulation when other programs are at an unknow point. The consequence of this problem is that the simulation results become less representative and less reproducible.
3) The task imbalance problem happens when the OS scheduler is involved when running simulations. Time spent on rebalancing a multi-program workload can be a major portion of the total simulation time. Consequently, the relative percentage of user-kernel cycles becomes non-deterministic.
4) Starting simulation from a potentially non-deterministic state results in higher simulation variation because each experiment starts from a different system state.

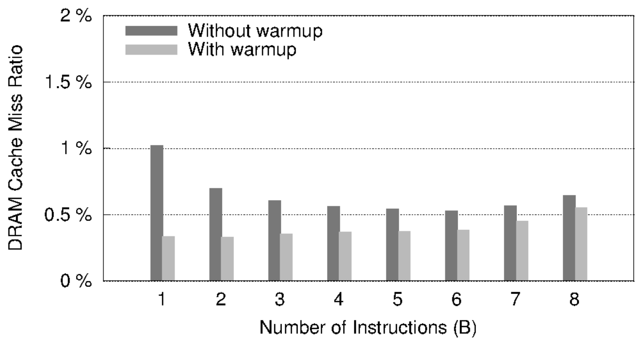
A. Cold Large Caches

To reduce the run time of simulations, a common practice is to truncate detailed execution of the workload. One method is to use a tool such as SimPoint [14] to get the RoI of a benchmark. Once the RoI is defined, we can fast-forward simulation until the starting point of the RoI is reached. Detailed simulation starts only after reaching the RoI, and then a certain number of instructions are executed, usually a few hundred million or few billion instructions. One problem with this simulation strategy is that, after fast-forwarding, the processor and memory states are still cold (invalid). The solution to this problem is to warm up the states before starting detailed simulation. For example, we can fast-forward X instructions, then run detailed simulation for Y + Z instructions but only track simulation statistics for Z instructions. Y in this case is the warmup period. When the cache is small, this method works since a few ten or hundred million instructions are sufficient to warm up the cache [28]. However, when the cache is large, such as the case of hybrid main memory architectures with a 1 GB DRAM cache [23], the number of instructions required for warmup would be prohibitive. For instance, as shown in Figure 1(a) where we compare the miss ratio of a 512 MB DRAM cache with and without warmup, we can see that even simulating 8 billion instructions, the cache miss ratio is still incorrectly inflated.

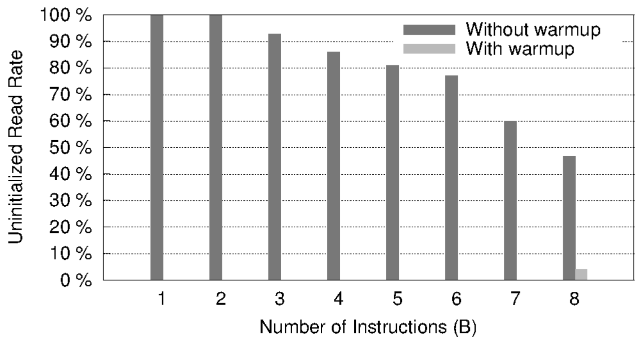
In addition to the cache miss ratio, we introduce the uninitialized read rate as another indicator of the degree to which the system has been warmed up. An uninitialized read refers to reading a memory page before the memory page is initialized by a write. In a real system, uninitialized read can only happen when the cache reads in a line from the backing store during the first write operation to that line since system boot. In general, the uninitialized read rate is low when the running application has reached its RoI. For instance, as shown in Figure 1(b) where we compare the uninitialized read rate of a 8 GB hybrid memory (a 512 MB DRAM cache and an 8 GB PCM main memory), the uninitialized read rate is over 40% even simulating 8 billion instructions after reaching the RoI. This indicates that the memory is incompletely warmed up. As a result, effective warmup is an important simulation step that is needed to ensure representative results. Furthermore, traditional approaches which consisted of executing a number of instructions prior to the RoI are not fully effective for architectures which incorporate a very large cache.

B. Non-representative Phase in Multi-Program Workloads

Evaluating multi-program workloads is more complicated than single program experiments, but multi-program simulations are essential for the exploration of currently ubiquitous multi-core architectures. There are studies which examine the methodologies for constructing workloads from program samples which subsequently simulate only those samples instead of simulating the complete program [16]. In a full-system simulation with multi-program workloads, one problem is where to start detailed simulation such that each individual program executes in its representative phase. In some setups, switching to detailed simulation happens at the RoI of the program that has the earliest RoI starting point. This is a problem because the other programs may still be in the initialization phase or in other non-representative phases. Figure 2 illustrates that each of the four constituent programs of the workload could potentially have their RoI at different points in time, and therefore starting detailed simulation from either the RoI of the first program or the RoI of the last program would not be representative. For instance, the simulation



(a)



(b)

Fig. 1. Without warmup versus complete warmup. (a) DRAM cache miss ratio. (b) Percentage of uninitialized read. An uninitialized read refers to reading an uninitialized memory page. An uninitialized page is a page that has never been accessed since system boot.

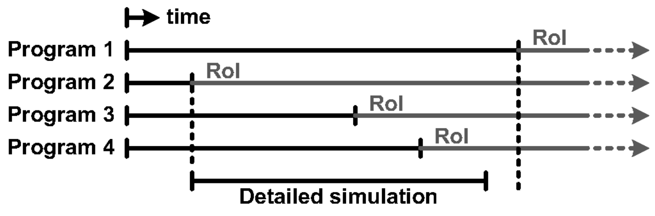


Fig. 2. RoI of each program starts at different point of time.

may finish without reaching the RoI of Program 1. In our simulation strategy, we have implemented a mechanism which synchronizes all programs to reach their RoI before starting the detailed simulation.

C. Workload Imbalance

In processor scheduling, there are separate run queues for each processor, and each processor only selects processes from its own queue to run. However, it is possible for one processor to be idle while others have processes waiting in their run queues. The scheduler thus rebalances the queues periodically: if one processor's run queue is too long, some processes are moved to another processor's queue.

Rebalancing the workload can happen in the range of hundreds of milliseconds on real systems [9], but in a full-system simulation environment, since at least 1000x slowdown is observed [22], it is common to simulate a few billion instructions, which is approximately equivalent to executing only 1 second of the workload on real hardware. As a result, if workloads are not balanced at the beginning of a simulation, it may take a large portion of the total simulation time for rebalancing to happen and the relative percentage of user-kernel cycles can be very different from run to run. Large variability can thus occur when measuring performance. Figure 3 illustrates two simulations, Sample 1 and Sample 2, with identical configurations but each having a different percentage of user cycles. This results in different IPC values.

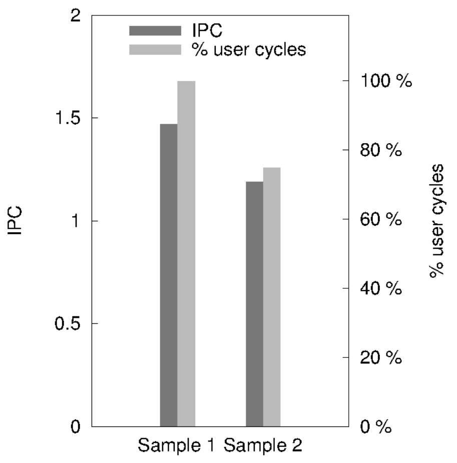


Fig. 3. Differences in percentage of user cycles and IPC of two simulations with identical configuration.

D. Non-deterministic Simulation Starting State

As described in section II-A, a common methodology to skip a workload's initialization phase is by fast-forwarding. Fast-forwarding constructs the architectural state of the starting point of the RoI by functional simulation from the beginning of the program. For example, the MARSSx86 simulator [22] runs in emulation mode using QEMU [8] prior to the RoI, then switches to detailed simulation mode once the RoI is reached [22]. However, fast-forwarding is non-deterministic and thus constructs different initial architecture states from run to run. Variation in initial architecture states can lead to different execution paths when running detailed simulation. For example, we are observing 20x more IPC variation, comparing non-fixed initial architecture state to fixed initial architectures state.

III. Simulation Framework and Workloads

A. Baseline Configuration

For our study, we use MARSSx86 [22], a full-system simulator for x86-64 CPUs. We also implemented a hybrid main memory system, which includes a DRAM cache [25], an NVM main memory system, and a hybrid cache controller (HybridSim) that manages both the DRAM and the NVM, as shown in Figure 4. MARSSx86 is based on QEMU, which is a dynamic binary translation system for emulating processor architectures. QEMU also emulates IO devices, chipsets, and other details, allowing it to boot unmodified operating systems. MARSSx86 relies on QEMU to emulate anything not directly

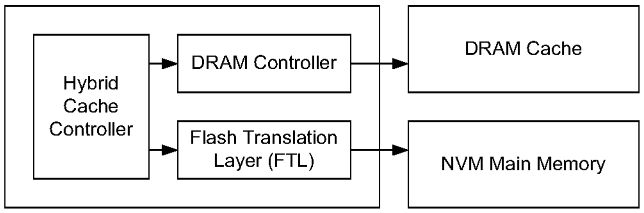


Fig. 4. Logical organization of a hybrid memory architecture.

TABLE I
BASELINE CONFIGURATION.

Processor	4-core, issue width = 4, 2 GHz
L1I (private)	32 KB, 4-way, 64 B block size
L1D (private)	32 KB, 4-way, 64 B block size
L2 (private)	256 KB, 8-way, 64 B block size
L3 (shared)	8 MB, 16-way, 64 B block size
DRAM cache	512 MB, 64-way, 4 KB page size
Non-volatile main memory	8 GB, 4 KB page size

executed by the detailed simulation. The baseline configuration is a quad-core, out-of-order system, with cache organization similar to the Intel Core i7. An 8 GB NVM is considered, with a 512 MB DRAM cache in front of it. The non-volatile DIMM organization has 1 channel, 2 dies per channel, 4 planes per die, 4096 blocks per plane, 64 pages per block, and each page is 4 KB. All transfers between the NVM and the DRAM occur at the page granularity. The timing parameters for the non-volatile memory are based on PCM numbers [18]. The DRAM cache, also in the form of a DIMM, is organized as 1 channel, 1 rank per channel, 8 banks per rank, 8192 rows per bank, and 1024 columns per row. All transfers between the DRAM and the L3 cache occur at the L3 cache line granularity (64 B). DRAM timing parameters are based on a Micron datasheet [26]. All devices are 8 bits wide. Table I summarizes our system configuration.

B. Workloads

We use 9 benchmarks to evaluate our system: mcf, milc, lbm, libquantum from the SPEC CPU 2006 suite [3]; canneal from the PARSEC 2.1 suite [10]; CG, FT, IS, MG from the NAS Parallel Benchmark 3.3.1 [2]. These benchmarks were chosen because they exhibit larger working set sizes and are more memory intensive. The input sets ref, large, and class B are used for the SPEC, PARSEC, and NAS benchmarks, respectively. For the SPEC and NAS benchmarks, we use SimPoint [14] to identify the RoI of each program. [1] Table II shows the characteristics of each benchmark. The L3 cache miss ratio is defined as the number of L3 cache misses divided by the number of L3 cache accesses (not all accesses that originate from the CPU core). The MPKI is the number of L3 cache misses per kilo-instructions. The working set size corresponds to the number of unique 4 KB pages touched in 250 million instructions executed starting from the RoI. We further form four-process multi-program workloads, as summarized in Table III. All workloads run on top of Ubuntu 9.04 (Linux 2.6.31).

[1]PARSEC has pre-defined RoI.

TABLE II
BENCHMARK CHARACTERISTICS.

Benchmark Suite	Benchmark	L3 Cache Miss Ratio	MPKI	Working Set Size (MB)
SPEC CPU 2006	mcf	34.4%	42.07	26.20
	milc	84.0%	1.39	329.06
	lbm	49.7%	18.64	424.02
	libquantum	50.3%	29.04	33.80
PARSEC	canneal	68.7%	25.95	171.87
NAS	CG	9.2%	18.56	168.74
	FT	6.8%	1.20	77.15
	IS	55.8%	10.51	278.37
	MG	36.0%	9.28	365.97

TABLE III
WORKLOAD SUMMARY

mix1	milc, lbm, canneal, MG	693 MB working set size
mix2	mcf, lbm, canneal, IS	674 MB working set size
mix3	milc, lbm, CG, FT	503 MB working set size
mix4	mcf, libquantum, CG, MG	465 MB working set size
lbmx4	4 copies of lbm	1047 MB working set size
milcx4	4 copies of milc	353 MB working set size

IV. SOLUTIONS

In order to avoid the simulation non-determinism presented in section II, we suggest four simulation techniques, which are *the detailed emulation replay warmup technique* that solves the cold large cache problem, using *RoI synchronization (RoI-sync)* to deal with simulating non-representative phase in multi-program workloads, setting *processor affinity* to solve the workload imbalance problem, and using *checkpointing* to avoid non-deterministic simulation starting state. These techniques are further described in detail in the following subsections.

A. Cold Large Caches

In order to be effectively warmed up, the large DRAM cache of a hybrid main memory system requires either a long warmup period or a replay of all the memory accesses before the detailed simulation begins. An accurate replay of the system recreates both the state of the DRAM cache and the NVM (mapping information, used bits, etc.). In addition, these states should capture both user-level memory accesses and OS boot-up accesses. Otherwise certain cache accesses might miss that should not. However, capturing all of this information would be time consuming if the state were recorded while the simulator was performing a detailed simulation. Therefore, instead of running detailed simulation from system boot, we track and save all of the last level cache transactions in a custom binary trace file from system boot to the starting point of the workload's RoI in QEMU's emulation mode. Subsequently, we translate the custom binary trace file from QEMU to an ASCII HybridSim-readable memory trace, and feed this memory trace to HybridSim using its stand-alone trace-based mode. HybridSim then outputs two files, one containing the DRAM cache state and the other containing the

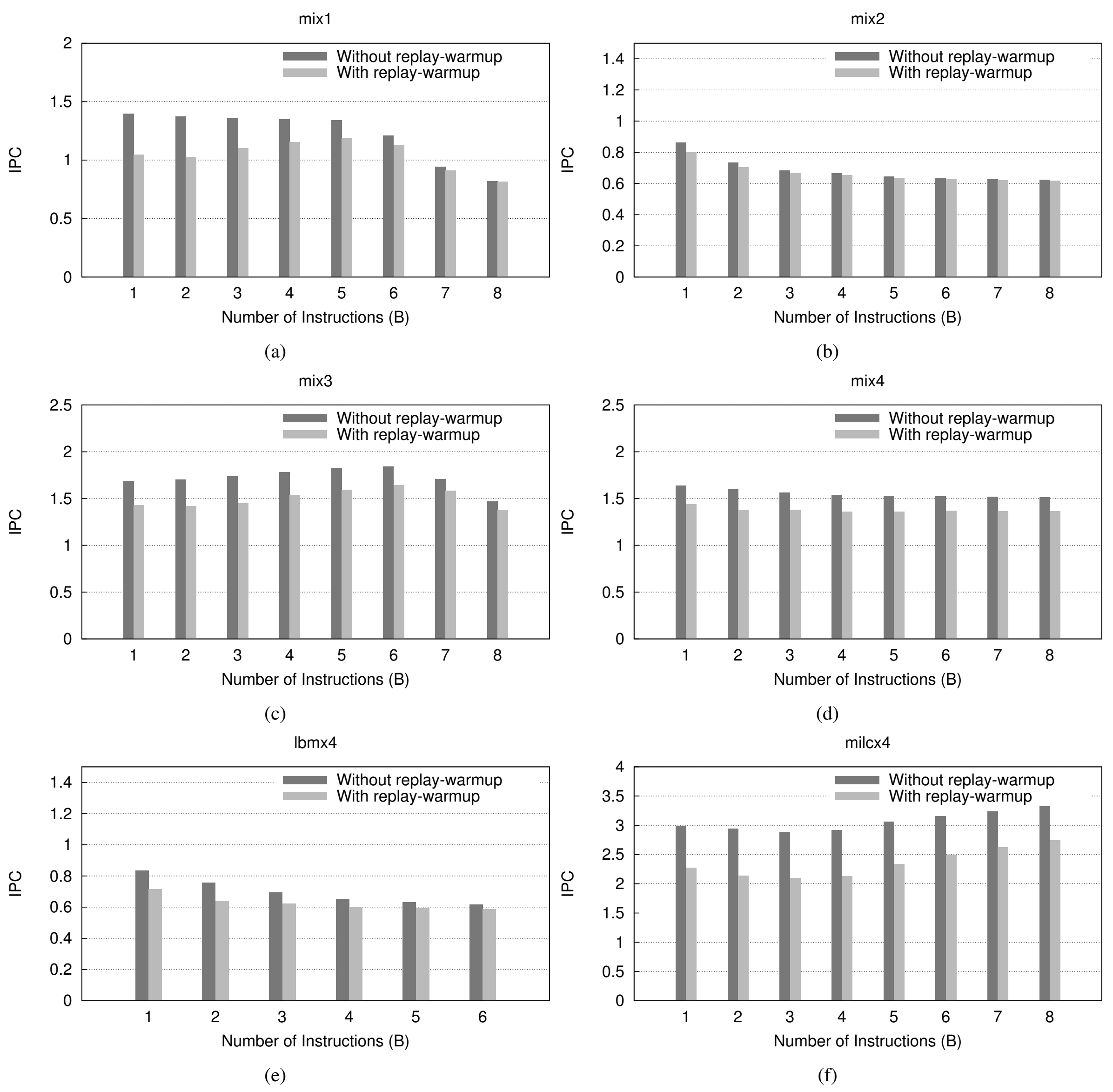
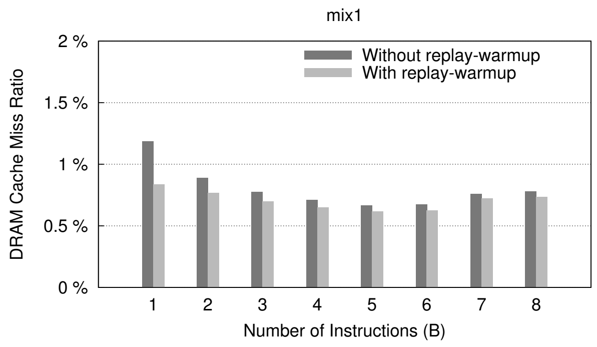


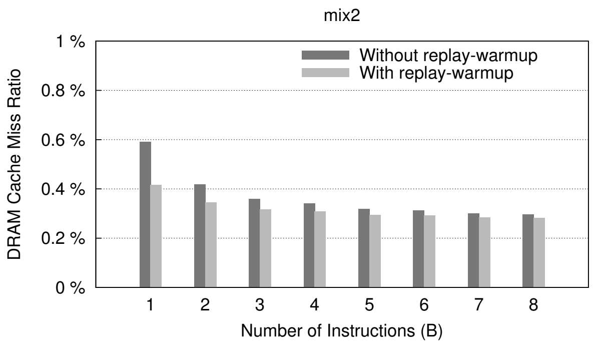
Fig. 6. Replay warmup IPC results. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

NVM state. We refer to this technique as *detailed emulation replay warmup*. The whole process of replay warmup takes around 2 hours for a typical workload. Since this process is accompanied by saving a checkpoint, the hybrid memory replay warmup state can be restored later with the checkpoint. This process is equivalent to a full simulation of all accesses from system boot to the RoI from the perspective of the hybrid main memory system. Figure 5 illustrates the flowchart of the replay warmup process.

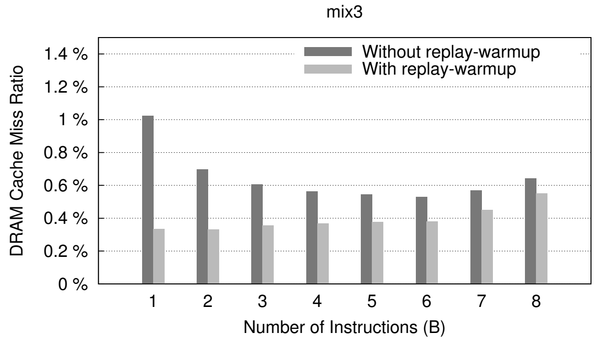
We compare results with replay warmup and without warmup. The other techniques in this paper are also applied for both cases. In particular, Figure 6 compares the IPC, Figure 7 compares the DRAM cache miss ratio, and Figure 8 compares the uninitialized read rate. The IPC values without warmup are larger than with replay warmup because the uninitialized reads in the NVM complete immediately. This is because no mapping exists for that page to a specific physical page in the NVM. We chose to return uninitialized reads immediately rather than fake the latency of the access as if it were mapped. We have run the simulations from 1 billion to 8 billion instructions and shown that if we do traditional warmup, the warmup period could take more than 8 billion instructions for some of our workloads. It is also shown in Figure 8 that with replay warmup, uninitialized read rates are zero for most cases, while without warmup, uninitialized read rates never drop below 20% even when running 8 billion instructions. In general, although we observe IPC and the DRAM cache miss ratio tend to converge when we run for a large number of instructions, a typical run of 8 billion instructions requires around 24 hours to complete. Thus, warming up for a few billion instructions creates a lot of overhead when running large numbers of experiments. Moreover, if we warm up for



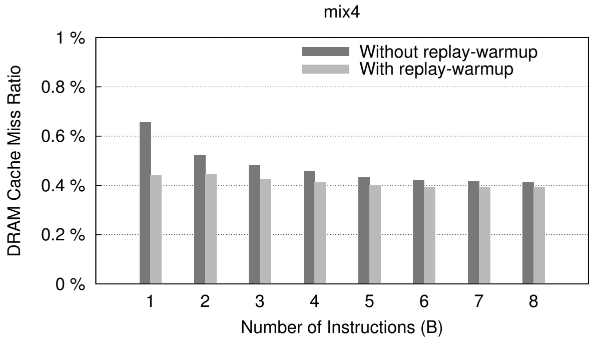
(a)



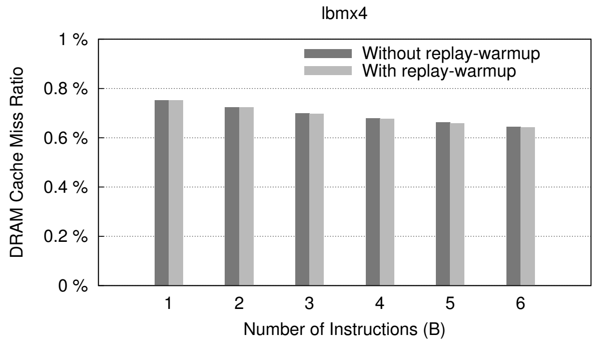
(b)



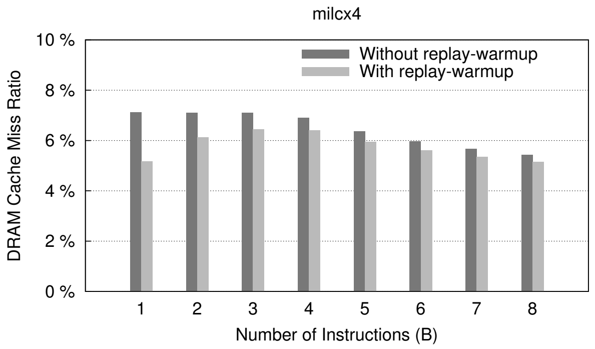
(c)



(d)



(e)



(f)

Fig. 7. Replay warmup DRAM cache miss ratio results. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

too long, then the statistics we capture no longer correspond to the representative RoI phase. For example, the mix1 workload shows a sudden decrease in IPC after 6 billion instructions complete because one of the benchmarks in the mix1 workload has changed its phase.

B. Non-representative Phase in Multi-Program Workloads

In multi-program workloads, capturing the simulation behavior of each program's representative phase is crucial. Therefore, one should make sure that when detailed simulation begins, each program starts from its RoI. To implement the RoI synchronization (RoI-sync) mechanism, we use System V interprocess communication semaphores [1] to communicate between different programs in a workload. The master process creates a semaphore with a unique key and sets the count of that semaphore to the number of programs in the workload. Then the master process forks a script into child processes which is comprised of specific commands to start the workload. Only this script needs to be changed when a different workload is used. Each individual program accesses the semaphore created by the master process at its RoI and uses the unique key to decrement the semaphore value by one. After decrementing the semaphore, each process then waits until the semaphore value reaches zero. In this way, the detailed simulation of all of the programs starts at the same time after all of the programs have reached their RoI.

Figure 9 shows the effects of using RoI-sync as opposed to starting detailed simulation when the first RoI is reached. Replay warmup and processor affinity are also utilized for both cases. We ran each experiment 20 times with 1 billion instructions per run. When not using the RoI-sync, the difference between the maximum and the minimum IPC normalized to the average IPC (which we refer to as the normalized max-

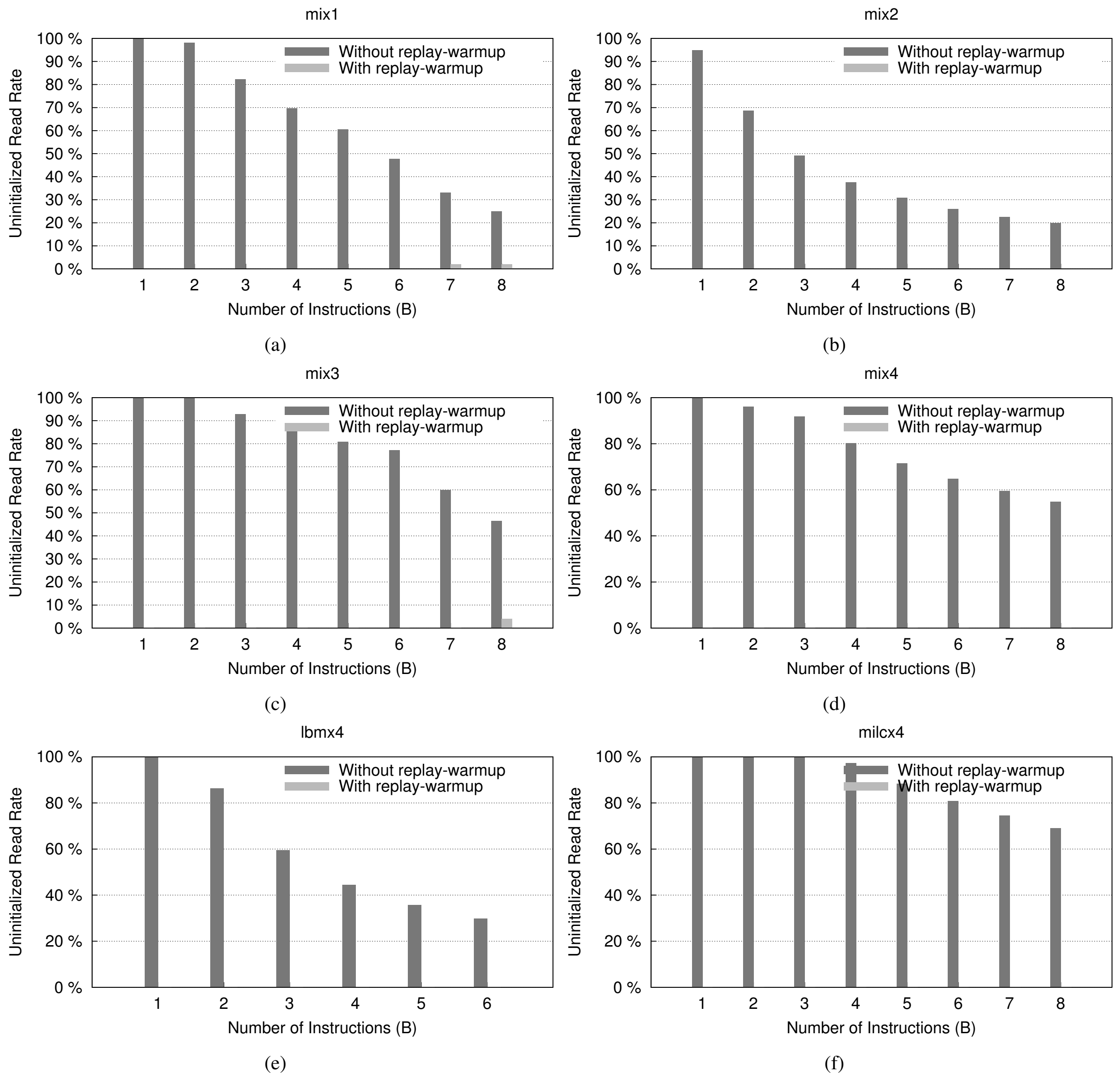


Fig. 8. Replay warmup uninitialized read rate results. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

imum variation of IPC) ranges from 30% to 183%. However, when RoI-sync is used, this value is between 5% to 14%. We observe more variability when running without RoI-Sync because when the earliest RoI is reached, the other programs in the workload are executing at non-deterministic locations which can vary from run to run. Note that even for the lbmx4 and milcx4 workloads, which consist of identical benchmarks and the same RoI, not synchronizing the starting point still results in larger variation. By using RoI-sync, it is ensured that each program starts from a deterministic simulation point and its representative phase is captured.

C. Workload Imbalance

The key to avoiding the workload imbalance problem is to make sure the workload is balanced before running detailed simulations, as opposed to allowing rebalancing of the workload during the detailed simulation period. Setting processor affinity is one possible method to achieve this. Processor affinity is a feature of the scheduling algorithm in a symmetric multiprocessing OS. Each process or thread has a tag indicating its preferred processors. At allocation time, each process or thread is scheduled to its preferred processors. On Linux, the CPU affinity of a process can be set or retrieved by the *taskset* program [19]. The CPU affinity is represented by a bitmask, in which the lowest order bit corresponds to the first logical processor and the highest order bit corresponds to the last logical processor. For example, assuming we are running four programs, programs 1 – 4 on a quad core architecture with processors 1 – 4, we can assign program 1 to processor 1, program 2 to processor 2, etc. as shown in Listing 1. Utilizing processor affinity can be a limitation when investigating OS scheduling. However, this technique effectively reduces non-determinism when targeting architectural research using full-system simulations.

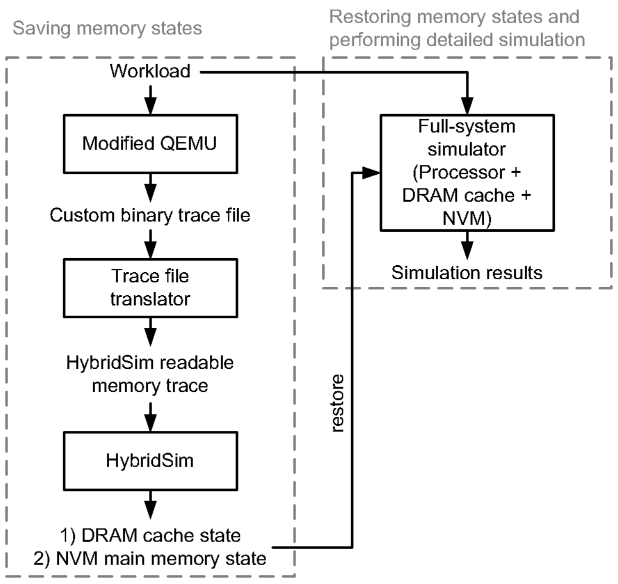


Fig. 5. Flowchart of replay warmup.

```
#!/bin/bash

taskset 0x1 [command to run program 1] &
taskset 0x2 [command to run program 2] &
taskset 0x4 [command to run program 3] &
taskset 0x8 [command to run program 4] &
```

Listing 1. Example of setting processor affinity by using taskset.

Figure 10 shows the variability reduction when using taskset. In each figure, there are two sets of data – without using taskset on the left and with using taskset on the right. Replay warmup and RoI-sync are also applied for both cases. For each workload, there is a box plot composed of four boxes. The darker boxes represent IPC and the lighter boxes represent user cycle percentage. We ran 20 samples for each experiment with 1 billion instructions per sample. Each box shows the minimum, the first quartile, the median, the third quartile, and the maximum for the metrics. When using taskset, user cycle percentages are very close to 100%. In addition, the largest difference between the maximum and minimum user cycle percentage is held to less than 0.1%. Using taskset also results in less IPC variation. On the other hand, when taskset is not applied, user cycle percentage can range from 24.9% to 99.8%, and it is shown that there is large variation in IPC. As a result, setting processor affinity can effectively reduce simulation variations and produce stable results.

D. Non-deterministic Simulation Starting State

Modern computer architecture simulators provide utilities to create and load checkpoints [11] [22] [20]. For example, MARSSx86 uses QEMU to create checkpoints in QCOW2 disk images and run simulations from the checkpoints. There are two advantages of using checkpoints. First, in full-system simulations, using checkpoints avoids time-consuming fast-forwarding from system boot. Second, checkpoints ensure

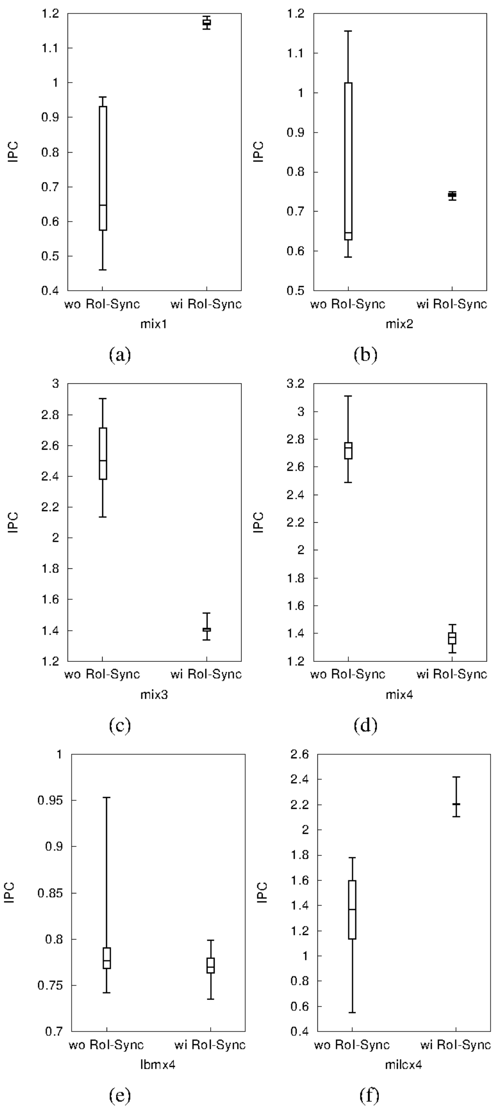


Fig. 9. The effect of using RoI-Sync. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

that the initial states are consistent for each simulation, thus reducing simulation variation.

However, there are disadvantages as well, which are addressed in [13]. In particular, checkpoints can require a large amount of storage space and loading a checkpoint can take a long time. These disadvantages are not critical in our simulation environment. The original disk image MARSSx86 provides, which is around 5 GB, is capable of storing checkpoints. In addition, the time overhead of loading checkpoints is small. For example, based on our simulation framework, it takes less than 1 minute to load a checkpoint, while it takes more than 2 minutes to fast-forward from system boot to the representative phase. Also, it takes around 3 hours for a 1 billion instruction simulation to complete, so the time to load a checkpoint is negligible.

Figure 11 shows the variability reduction when using checkpoints. The other techniques in this paper are also applied for

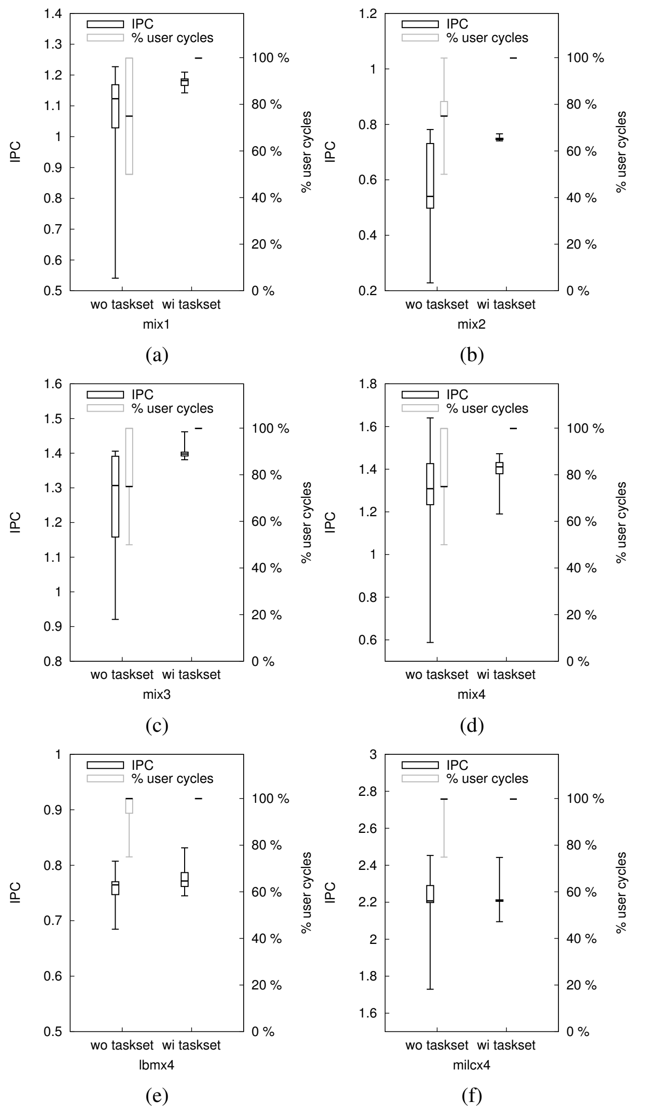


Fig. 10. The effect of using taskset. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

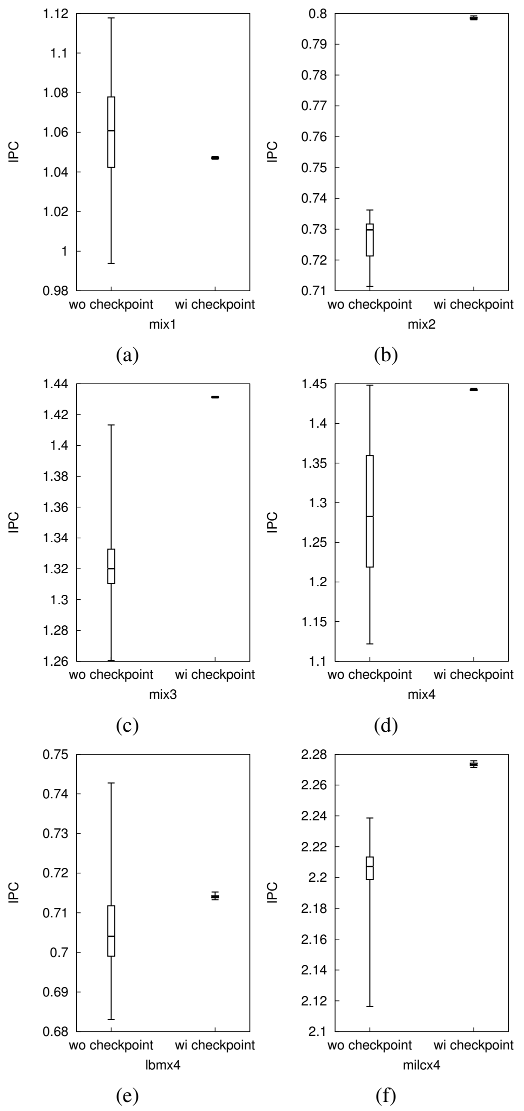


Fig. 11. The effect of using checkpoint. (a) mix1. (b) mix2. (c) mix3. (d) mix4. (e) lbmx4. (f) milcx4.

both cases. We ran 20 samples for each experiment with 1 billion instructions per sample. When not using checkpoints, the normalized maximum variation of IPC ranges from 3% (mix1) to 25% (mix4). But when using checkpoints, the normalized maximum variation of IPC for all workloads is less than 1%. It is thus shown that simulation variations can be effectively controlled by using checkpoints.

V. Related Work

Simulation has been the major performance evaluation method for computer architecture research. Eeckhout [13] provides an overview of the current state of computer architecture performance evaluation methods. In particular, chapter 4 presents multi-program workloads as a performance metric, chapter 5 reviews full-system simulation, and chapter 6 presents techniques on how to initialize architecture state when using sampled simulation. Yi and Lilja [29] also give a summary of simulation of computer architectures. In addition, simulation variability has been studied in the literature. Alameldeen and Wood [5] present the variability phenomenon in architecture simulations of multi-threaded workloads and how it can lead to wrong conclusions. Their work builds on [4] which identified commercial workload variability. They also demonstrate that, due to non-deterministic multi-threaded simulations, IPC as a performance indicator can be misleading [6]. Heirman et al. [15] specifically characterize and analyze variability in scientific parallel applications. Mytkowicz et al. [21] describes measurement bias in computer architecture research, where small changes in experimental setup can result in overstated or incorrect conclusions. Burugula and Skadron [12] proposed profiling user applications offline to get the memory reference reuse latency. The memory reference reuse latency further facilitates fast warmup for sampled microarchitecture simulations.

This paper focuses on producing reliable and repeatable results in full-system simulation of CMP computer systems with large caches. Systems with large caches have become more common. However, none of the existing works demonstrate techniques that warmup large caches efficiently. Moreover, to the best of our knowledge, none of the existing works show the use of interprocess communication for forming RoI synchronized multi-program workloads. In addition, we show the effectiveness of using processor affinity, and checkpointing, where the simulation variability is greatly reduced in a multicore full-system simulation environment.

VI. Conclusions

In this paper, we present techniques to produce reliable results in full-system simulation of CMP with large caches. We propose two techniques: detailed emulation replay warmup, which prevents simulation inaccuracies due to large cold caches, and RoI synchronization, which prevents simulating non-representative phase when running multi-program workloads. Additionally, we quantify the use of processor affinity and checkpointing. By utilizing all of these techniques, it is shown that simulation variability is effectively controlled and simulation results become more reliable and consistent. These simulation techniques will help prevent designers using full-system simulators from drawing incorrect conclusions.

References

[1] Beej's Guide to Unix IPC.

[2] NAS Parallel Benchmarks.

[3] SPEC CPU 2006.

[4] A. R. Alameldeen, C. J. Mauer, M. Xu, P. J. Harper, M. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood. Evaluating Non-deterministic Multi-threaded Commercial Workloads. In *Proc. Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2002.

[5] A. R. Alameldeen and D. A. Wood. Variability in Architectural Simulations of Multi-threaded Workloads. In *Proc. HPCA*, 2003.

[6] A. R. Alameldeen and D. A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *IEEE Micro*, 26(4):8–17, Jul.-Aug. 2006.

[7] M. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, Chi-Keung Luk, G. Lyons, H. Patil, and A. Tal. Analyzing Parallel Programs with Pin. *IEEE Computer*, 43(3):34–41, Mar. 2010.

[8] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proc. ATEC*, 2005.

[9] S. M. Bellovin.

[10] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, Jan. 2011.

[11] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, Jul.-Aug. 2006.

[12] R. S. Burugula and K. Skadron. Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation. In *ISPASS*, 2003.

[13] L. Eeckhout. *Computer Architecture Performance Evaluation Methods*. Morgan & Claypool, 2010.

[14] G. Hamerly, E. Perelman, J. Lau, and B. Calder. SimPoint 3.0: Faster and More Flexible Program Analysis. In *Proc. Workshop on Modeling, Benchmarking and Simulation*, 2005.

[15] W. Heirman, J. Dambre, D. Stroobandt, and J. Van Campenhout. Runtime Variability in Scientific Parallel Applications. In *Proc. Workshop on Modeling, Benchmarking and Simulation*, 2008.

[16] A. Hilton, N. Eswaran, and A. Roth. FIESTA: A Sample-Balanced Multi-Program Workload Methodology. In *Proc. Workshop on Modeling, Benchmarking and Simulation*, 2009.

[17] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro*, 30(2):7–15, Mar.-Apr. 2010.

[18] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proc. ISCA*, 2009.

[19] R. M. Love. Linux users manual.

[20] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.

[21] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing Wrong Data Without Doing Anything Obviously Wrong! In *Proc. ASPLOS*, 2009.

[22] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Proc. DAC*, 2011.

[23] M. K. Qureshi, V. Srinivasan, and J. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proc. ISCA*, 2009.

[24] A. Rico, A. Duran, F. Cabarcas, Y. Etsion, A. Ramirez, and M. Valero. Trace-Driven Simulation of Multithreaded Applications. In *Proc. ISPASS*, 2011.

[25] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1):16–19, Jan. 2011.

[26] Micron Technology. DDR3 SDRAM, 2010.

[27] Yun W, K. Kang, and C.-M. Kyung. Thermal-Aware Energy Minimization of 3D-Stacked L3 Cache with Error Rate Limitation. In *ISCAS*, 2011.

[28] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins. Characterizing and Comparing Prevailing Simulation Techniques. In *Proc. HPCA*, 2005.

[29] J. J. Yi and D. J. Lilja. Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations. *IEEE Trans. Computers*, (3):268–280, Mar. 2006.