10-2016

A Comparison of x86 Computer Architecture Simulators

Ayaz Akram
Western Michigan University, ayaz.akram@wmich.edu

Lina Sawalha
Western Michigan University, lina.sawalha@wmich.edu

A Comparison of x86 Computer Architecture Simulators

Ayaz Akram and Lina Sawalha
Electrical and Computer Engineering Department
Western Michigan University, Kalamazoo, MI 49008

Abstract

The significance of computer architecture simulators in advancing computer architecture research is widely acknowledged. Computer architects have developed numerous simulators in the past few decades and their number continues to rise. This paper explores different simulation techniques and surveys many simulators. Comparing simulators with each other and validating their correctness has been a challenging task. In this paper, we compare and contrast x86 simulators in terms of flexibility, level of details, user friendliness and simulation models. In addition, we measure the experimental error and compare the speed of four contemporary x86 simulators: gem5, Sniper, Multi2sim and PTLsim. We also discuss the strengths and limitations of the different simulators. We believe that this paper provide insights into different simulation strategies and aims to help computer architects understand the differences among the existing simulation tools.

1 Introduction

Due to huge costs associated with building real systems for testing and verification purposes, computer architects rely on simulation and modeling techniques to evaluate different design options. Major percentage (approximately 90%) of papers published in top conferences use simulation as performance evaluation methodology [1]. Previous surveys on computer architecture simulations are old and do not include recent simulators [2]. Some of them focus only on teaching or memory related simulators [3, 4]. This paper compares and contrasts x86 simulators based on different characteristics and features, and show several examples of contemporary simulators. x86 is one of the oldest and widely used instruction set architectures (ISAs) in desktops and servers. With the recent diversion of x86 towards mobile computing, it is gaining more attention and use.

There is not much literature dealing with the evaluation of x86 simulators by comparing them to each other and to the state-of-the-art processors. The absence of performance validation of simulators may cause experimental errors that can result in incorrect conclusions, if the errors are large. Nowatzki et al [5] discussed some reasons for why finding and correcting errors in simulators is difficult including: implicit assumption of simulator features, absence of detailed documentation, inefficient and mislabeled microoperations that causes unnecessary register dependencies, etc. Academic researchers often use experimental error of a simulator's ability to model an existing hardware [6, 7]. In this paper, we configure four state-of-the-art x86 computer architecture simulators to model Intel's Haswell microarchitecture. Then we quantify the experimental errors and evaluate the accuracy of such an approach. In addition, we compare the performance results of the simulators among each other, and pinpoint some

causes of inaccuracies. The paper also discusses the strengths and limitations of different x86 simulators. The purpose of this paper is not to criticize a particular simulator for showing high experimental error, but to emphasize the importance of validating simulators and to help the community understand some sources of inaccuracies.

Our main contributions in this paper are:

- An up-to-date survey of various types of simulators that support x86 ISA with a comparison of their features and characteristics.
- A detailed comparison of four modern x86 simulators: gem5 [8], Sniper [7], Multi2sim [9] and PTLsim [10].
- A comparison of simulation speed of these four simulators, in addition to quantifying the experimental errors of each simulator modeling real hardware.

The rest of this paper is organized as follows: section 2 discusses simulators' categories and examples of x86 simulators that belong to each category. Section 3 describes in detail four contemporary x86 simulators. Section 4 discusses our experiments methodology to verify the performance of the simulators and measure the experimental error, and section 5 shows the obtained results and analysis of simulators verification. Finally, section 6 concludes the paper.

2 Classification and Survey of Simulators

Simulators can be divided into various types based on: the level of simulation details, the scope of target, and the input to the simulator. Next, we discuss the classes of x86 simulators based on the aforementioned factors.

2.1 Functional vs. Timing Simulators

Functional simulators simulate the functionality of the target only and do not model microarchitectural details. Examples of functional simulators that support x86 are 'sim-safe' and 'sim-fast' models of SimpleScalar [11] and Simics [12].

Timing simulators (also referred as performance simulators) provide detailed information about the real timing/performance of a simulated computer system or parts of a computer system. Timing simulators that keep track of all clock cycles on a simulated processor while a particular benchmark is executing on it, are known as cycle-level simulators. Keeping track of such detailed information makes them slow and resource intensive in comparison to functional simulators and other subtypes of timing simulators [13]. For example, the fastest functional simulator for SimpleScalar can simulate instructions 25 times faster than its detailed (cycle-level) simulation model known as 'sim-outorder'. Other examples of cycle-level simulators are GEMS [14], Gem5 [8], PTLsim [10] and Flexus [15].

Event-driven simulators perform simulation corresponding to events, instead of simulating the target on a cycle-by-cycle basis (thus save simulation time). It is also possible to combine cycle-level to event-driven simulators by modeling some parts of the simulated structures on a cycle-by-cycle basis and others on an event-driven basis. McSimA+ [16] is a timing simulator that supports both in-order (IO) and out-of-order (OOO) x86 pipelines. SimFlex [17], which is based on sampling microarchitecture simulation (SMARTS) [18] framework, and Flexus (Simics) [15] are cycle-level timing simulators while some components are event-driven internally. Many alternatives to cycle-level and event-driven simulators have emerged recently,

which provide a good tradeoff between simulation speed and accuracy. For example, interval simulation [19] has been recently proposed, which focuses on simulation of multi-core systems. The fundamental idea of interval simulation is that the miss events like branch mispredictions and cache misses divide the normal flow of instructions through the pipeline into intervals. These miss events can be determined by respective simulator blocks (e.g. branch predictor simulator, memory simulator) and an analytical model can be used to find the timing for each interval. Using both the miss event simulators and mechanistic analytical model, performance information can be acquired. Sniper [7], an x86 multicore simulator, is based on this technique.

2.2 Application-Level vs. Full-System Simulators:

Application-level/User-mode simulators are able to run only target applications instead of a full fledge target operating system (OS). Thus, they have to simulate microprocessor and only limited peripherals. In user level simulations, any system service requests by the simulated application/benchmark bypass the user-level simulation and are serviced by the underlying host OS. For compute intensive benchmarks for example SPEC CPU, little time is spent in the execution of system-level code [20], thus relying only on the simulation of user-level code might not be a problem. However, for many other workloads (server workloads, transaction processing and database benchmarks), this simulation will not be enough because a considerable amount of time is spent in the execution of system-level code. Application-level simulators are usually less complex but may show some inaccuracies due to lack of system level code support. SimpleScalar, Sniper [7] and McSimA+ [16] are examples of application-level simulators. Zesto [21] (built on top of SimpleScalar) and ZSim [22] are application level timing simulators that support x86 ISA.

Full system simulators can simulate an entire OS for the target. I/O devices needed to boot an OS are also simulated. Gem5 [8], ML-RSim [23], MARSSx86 [24] and PTLSim [10] are some examples of full system timing simulators.

2.3 Trace-Driven vs. Execution-Driven Simulators:

Trace-driven simulators use trace files as their inputs. These files contain recorded streams of instructions from a program's run on some real hardware. These simulators do not need to emulate the ISA of the target, which makes them relatively simple. One problem with the trace-driven simulators is that the generated trace files can be very large in size and reading such large files from disk can be slow. This can be solved by using trace sampling and reduction techniques [25].

In the case of execution-driven simulation, instructions of any particular benchmark are executed on the simulated machine directly. Different performance related aspects are calculated at the same time as the program is executing. As opposed to trace-driven simulators, these simulators can simulate misspeculated code paths. However, they are more complex compared to trace-driven simulators and can take longer to run if statistical sampling is used to simulate portions of the benchmarks because of fast forwarding time. SimpleScalar [11], Augmint [26] and Gem5 [8] are examples of these simulators.

There are four main factors to consider when comparing different computer architecture simulators: performance, flexibility, detail of simulation model and accuracy. Table 1 shows a comparison of some detail and flexibility features of several x86 computer architecture simulators. In the next sections, we compare four of these simulators with more details.

Table 1: Comparison of x86 Simulators

Simulator	Supported hosts (ISA/OS)	Category	Supported Processor Models	MultiCore Support
Gem5	x86, ARM, SPARC, Alpha, PPC/Linux,MacOSx,Solaris,OpenBSD	FS/MOD/TIM (cycle-accuate)	IO, OOO	yes (HMP)
Multi-2sim	x86/Linux	UM/MOD/TIM	OOO, multithreaded	yes (HMP)
Sniper	x86/Linux	parallel UM/TIM (interval simulation)	IO, OOO, SMT cores	yes (HMP)
PTLsim	x86/Linux	FS/TIM simulator (cycle-accurate)	OOO	yes
ZSim	x86/Linux	parallel UM/TIM	IO, OOO	yes (HMP)
Simple-Scalar	x86/Linux, Win2000, SPARC/Solaris	UM/ED/TIM	OOO	no
SIMFLEX	x86	FS/MOD/TIM (decoupled functional and timing)	OOO	yes
SIMICS	Alpha, PPC, UltraSparc, x86/Linux, Windows	FS/functional	functional	yes
Augmint	x86/Unix and Windows NT	ED/TD	functional	yes
Mc-SimA+	x86/Linux	UM/TIM (decoupled functional and timing)	IO, OOO	yes (HMP)
GEMS	x86/Linux, AMD64-linux, and SparcV9 (Solaris 8)	FS/TIM (decoupled functional and timing)	OOO	yes
MARSSx86	x86-64/Linux	FS/TIM	IO, OOO	yes
Graphite	x86/Linux	parallel UM/TIM (decoupled)	IO	yes
OVPsim	x86/Windows and Linux	functional	IO	yes
Flexus	x86/Linux	FS/TIM/ED (cycle-accurate)	IO, OOO	yes
Zesto	x86	UM/TIM cycle-level	OOO	yes
McPAT	x86/Linux	power, area and TIM	IO, OOO, NOCs	yes (HMP)
Note: UM=user mode, **FS**=full-system, **ED**=execution-driven, **EvD**=event-driven, **TD**=trace-driven, **TIM**=timing, **MOD**=modular, **IO**=in-order, **OOO**=out-of-order, **HMP**=hetrogeneous multiprocessor				

3 Selected Simulators for Detailed Study

We selected four simulators for detailed study, *gem5* [8], *Multi2Sim* [27], *Sniper* [7] and *PTLsim* [10], because they have diverse design strategies with respect to detail and abstraction. All of them are contemporary simulators with active development, except PTLsim that is currently not in active development but still being heavily used. Next, is a brief discussion of all these simulators along with validation efforts that were performed for them in the past.

3.1 Gem5

Gem5 [8] is a full system simulator that supports many ISAs with various CPU models (Table 1). Gem5 borrows the detailed CPU modeling from M5 [28] and the detailed memory system modeling from GEMS [14]. Gem5 primarily supports four CPU models: 'AtomicSimple', 'TimingSimple', 'InOrder' and 'O3'. AtomicSimple and TimingSimple are non-pipelined single-cycle microarchitectures. The 'InOrder' and 'O3' are pipelined IO and OOO core models. Both are 'execute-in-execute' meaning that instructions are only executed in the execute stage after all dependencies have been resolved. These can be configured to simulate different number of pipeline stages, issue widths and number of hardware threads.

Gutierrez et al. [29] and Butko et al. [30] validated the accuracy of gem5 by modeling real systems based on ARM. After making some modifications to the simulator, apart from configuring it to match the experimental board (ARM Cortex A15), Gutierrez et al [29] were able to achieve a mean percentage runtime error of 5% and a mean absolute percentage runtime error of 13% for SPEC CPU2006 benchmarks [29]. Butko et al. [30] have analyzed the accuracy of gem5 for simulation of a multicore embedded system (ARM Cortex A9). Various benchmarks related to scientific workloads (SPLASH-2), media applications (ALPBench) and memory bandwidth (STREAM) were used for validation. The results show that the accuracy varies form 1.39% to 17.94%. We have not been able to find any validation effort for x86 based targets for gem5 simulator.

3.2 Sniper

Sniper [7] is a parallel simulator for simulating large scale multicore systems using interval simulation [19], which provides a balance between detailed cycle-level simulation and one-IPC simulation model (one-IPC model is defined as an IO single issue pipeline model). Sniper

is based on Graphite [31] that supports various one-IPC models. Carlson et al [7] validated Sniper against real hardware (4-socket 6-core Intel Xeon X7460 Dunnigton shared-memory with simultaneous multithreading (SMT) support) using SPLASH-2 benchmark suite. They concluded that Sniper's interval simulation is within 25% accuracy on average compared to real hardware. Carlson et al. introduced the instruction-window centric core model that improved the accuracy of base interval simulation model [32]. Validation of this model against Nehalem microarchitecture based system showed a single-core error of 11.1% using a subset of SPLASH-2 benchmarks.

3.3 Multi2Sim

Multi2Sim is another recently developed simulator for CPU-GPU computing, which supports various ISAs (Table 1). Multi2Sim [9] uses three different simulation models: functional, detailed and event-driven. The detailed simulator and the event-driven module collectively perform timing simulation. Multi2sim supports only OOO execution mode, and can support SMT. Memory hierarchy and interconnect networks are highly configurable. Multi2Sim borrows some of its very basic modules from Simplescalar [11]. It does not simulate an OS but still can execute parallel workloads with dynamic threads creation. Ubal et al [27] validated Multi2Sim for GPU simulation using commercial AMD Radeon 5870 GPU as a reference model, and AMD OpenCL SDK [33] as benchamark inputs. They verified functional correctness of the GPU simulator. The average error percentage between the execution time and simulated execution time vary from 5% to 30%. We have not found any validation efforts for x86 CPUs on Multi2Sim.

3.4 PTLsim

The fourth simulator that we experimented with is PTLsim, a cycle-level full-system x86 simulator. It can model a superscalar OOO core with SMT and a simple non-pipelined sequential core which is designed for functional simulation only. The default pipeline model is a modern OOO, based on a combination of features from the Intel Pentium 4, AMD K8, Intel Core 2, IBM Power4/Power5 and Alpha EV8 [10].

Yourst [10] configured PTLsim processor pipeline and memory hierarchy as close as possible to real 2.2 GHz AMD Athlon 64 (K8 micro-architecture) based machine, for experimental evaluation. He used *rsync*, client server benchmark to include kernel-level effects in simulation. By comparing PTLsim statistics and the real K8-based core's performance counters, the author measured a 5% error compared to the real hardware run.

Table 2 shows a comparison among different features and configuration parameters of the aforementioned four simulators in addition to a recently developed simulator ZSim [22]. gem5 supports the highest number of OSes and architectures as shown in Table 1. Sniper runs on any modern Linux-OS. Multi2Sim supports Linux, MacOS X (X86, ppc). PTLsim runs on Linux based x86 or x86-64 machine.

Fast forwarding and cache warmup are supported by gem5 and Sniper. Multi2Sim and ZSim support only fast-forwarding. PTLsim does not have a working fast forwarding or warmup. Sniper cannot create checkpoints itself, but it can use checkpoints created by Pin [34] and Simpoint tools. Only gem5 and Multi2Sim support the creation and later use of checkpoints during simulation. All of these simulators support the creation of execution traces during simulation. Gem5 produces very detailed simulated performance statistics (including pipeline stages throughput, instruction mix and performance of various structures). Multi2Sim and PTLsim

Table 2: Feature Comparison

Feature	Gem5	Sniper	PTLsim	Multi2Sim	ZSim
Platform support	P++	P	P	P+	P
Target support	T++	T	T	T+	T
Full system	✓	X	✓	X	X
Fast forwarding & cache warmup	✓	✓	X	✓	✓
Checkpointing	✓	X	X	✓	X
Trace generation	✓	✓	✓	✓	✓
Details of generated performance stats.	D++	D	D+	D+	D+
Pipeline depth configuration	✓	X	✓	X	✓
Energy and power modeling	E++	E	E	E-	E
In-order pipeline support	✓	✓	X	X	✓
HMP support	M,G,S	S	X	M,G	S
GPU-Modelling	✓	X	X	✓	X
Multi-threaded app. support	✓	✓	✓	✓	✓
Community support	C++	C++	C-	C	C+

Note: [feature's 1st letter]++ is better than [Feature's 1st letter]+ which is better than [feature's 1st letter] which is better than [Feature's 1st letter]- , S=Single-ISA, M=Multi-ISA, G=GPU

also produce detailed statistics but the details are less than those produced by gem5. Sniper does not produce very detailed performance statistics. In Gem5, PTLsim and ZSim, branch misprediction penalty can be configured implicitly by varying the pipeline depth. Sniper and Multi2Sim do not have explicit options to configure pipeline depth, but they have options to set misprediction penalty and instruction latencies. To support power and energy modeling, gem5 and Sniper provide support for dynamic voltage and frequency scaling (DVFS), making it possible to run experiments to simulate energy efficiency. Multi2Sim, Gem5, Sniper and ZSim can interact with McPAT [35] to evaluate power and energy consumption. PTLsim does not provide any support for energy consumption modeling. In terms of the support for heterogeneous multicore (HMP) simulation, Sniper and ZSim support only single-ISA (also known as asymmetric) HMP simulation. Gem5 has currently integrated a GPUsim model to run cpu-gpu heterogeneous simulations. Moreover, it can be modified to simulate single- and multi-ISA HMP system, as it supports multiple ISAs. Multi2Sim integrates models for CPU and different GPU architectures and has extensive support for GPU-CPU computing simulation. Gem5 and Sniper have large development communities and support forums. Multi2Sim and ZSim also maintain support groups to discuss problems related to the simulators. There is currently no maintained support forum for PTLsim.

4 Simulators Verification Methodology

To experimentally evaluate the error in the selected simulators to model real hardware, we have configured them to model an existing processor, *Corei*7. The target system is similar to Haswell μ-architecture (Intel core i7 machine i7-4770 cpu, 3.40 GHz). We then compare the performance of simulated benchmarks with their runs on the real hardware. This section discusses the configurations used for the simulators, experimental workloads and the method followed for measuring real hardware metrics.

4.1 Target System (Core i7 Like)

As all the exact configurations for this processor are not published by Intel, we tried our best to model Haswell microarchitecture based on some Intel documentation [36, 37] and other resources [38, 39, 40, 41]. The basic features of this target system are in Table 3. The simulators are configured to model a 19-stage pipeline with 14-cycles misprediction penalty. All simulators in this study do not model fused μ-ops, thus we use pipeline stages' widths in equivalent μ-ops. As we do not have the exact specifications of branch predictors in Haswell μ-architecture, and not all simulators implement the same branch predictors, we have tried to configure similar tournament branch predictors in all simulators. We use a 4K-entries branch target buffer (BTB) and 16-entries return address stack (RAS) with a tournament predictor where at least one of them is a 2-level predictor. Details of these predictors are available in next sub-sections.

Table 3: Target Configurations.

Parameter	**Core i7 Like**
Pipeline	Out of Order
Pipeline stages	19
Fetch width	6 instructions per cycle
Decode width	4-7 fused μ-ops per cycle
Decode queue	56 μ-ops
Rename width and Issue width	4 fused μ-ops per cycle
Dispatch width	8 μ-ops per cycle
Commit width	4 fused μ-ops per cycle
Reservation station	60 entries
Reorder buffer	192 entries
Number of stages	19
L1 data cache	32KB, 8 way
L1 instruction cache	32KB, 8 way
L2 cache size & Associativity	256KB, 8 way
L3 cache size & Associativity	8 MB, 16 way
Cache line size	64 Bytes
L1 cache latency	4 cycles
L2 cache latency	12 cycles
L3 cache latency	36 cycles
Instruction latencies	Based on [36, 38]
Branch target buffer	4096, 4 way
Return Address Stack	16 entries
Branch misprediction penalty	14 cycles
Physical Int/FP registers	168 each
Instruction TLB	128 entries
Data TLB	64 entries, 4 way
L2 TLB	1024 entries, 8 way

4.2 Configurations of Simulators for Core i7 Target

Our main objective while configuring simulators was to be as close as possible to the real target system. Another, important aim was to configure these simulators as close as possible to each other, so that we could perform a fair and impartial comparison. Although these simulators have diverse support for configuration parameters, we tried our best to keep the simulation configurations the same. We also performed initial level accuracy experiments with ZSim. This section includes ZSim configurations as well, but we do not include results from ZSim experiments in this report. Table 4 shows differences in configurations across all simulators due to different level of support for different features by these simulators. Table 5 shows detailed configurations for all simualtors. First column describes the configuration parameter, while rest of the columns describe the corresponding simulator's parameter name and set value for this parameter. Definition of that parameter for a particular simulator is also mentioned in brackets where required.

Table 4: Differences in Simulator Configurations

Parameter	**Gem5**	**Sniper**	**PTLsim**	**Multi2Sim**	**ZSim**
Branch Predictor	Tournament (Local:4K, Gobal: 4K,Choice: 4K)	Pentium M (Bimodal:4K, global:4K)	Hybrid (Bimodal: 4K, Gshare: 4K, Choice: 4K)	Combined (Bimodal :4K,2 Level: 4K,Choice:4K)	2-level BP (L1 size:4K, L2 size: 4K)
BTB associativity	1 way (direct)	4 way	4 way	4 way	ideal btb
TLB associativity	1 way (direct)	4, 8 way	1 way	-	no tlb
Instruction queue	unified (60 entries)	unified (60 entries)	distributed (4 queues, 16 entries each)	unified (60 entries)	unified (60 entries)
Memory address disambiguation	store sets	NC	NC	NC	NC
Delay between pipeline stages	configured to achieve overall 19 stage pipeline	NC	NC	NC	configured
TLB levels	single	double	single	-	no tlb
Note: NC = Not Configurable					

Gem5's pipeline configuration is very flexible, in fact it is the most flexible out of the four simulators that we have studied. A 19-stage pipeline is set by configuring different delay values between the existing pipeline stages of gem5's OOO pipeline, such that branch misprediction penalty stays 14 cycles. Since μ-ops are known at the fetch stage in gem5, all pipeline width parameters are set in μ-ops. Moreover, to incorporate the effect of fused μ-ops (not supported in gem5), we have translated the width of 4 fused μ-ops to 6 normal μ-ops. The branch predictor simulated is a tournament predictor similar to the one present in Alpha 21264 machine [42]. This scheme maintains local and global history to predict the direction of a given branch, where a choice predictor selects one out of the two predictions. The size of local, global and choice predictors and direct-mapped BTB are 4K entries each. Each entry contains a 2-bit counter.

In case of Sniper, to configure any target system, a primary configuration script *base.cfg* is already provided. Various scripts used to configure different core models and structures are available that can override the default configuration parameters. Moreover, configurations for some modern Intel processors are also provided. We have overwritten the basic configurations with those for Haswell.

Table 5: Simulators Configuration Table

Parameter	**Gem5 Parameter**	**Multi2Sim Parameter**	**PTLsim Parameter**	**Sniper Parameter**	**ZSim Parameter**
Clock frequency of core	cpu-clock = 3.4GHz	Frequency = 3400 MHz	NP	frequency = 3.4 GHz	frequency = 3400 MHz
Core model	cpu-type = detailed (out of order pipeline)	x86-sim = detailed (out of order pipeline for x86 cpu)	ooo core (out of order pipeline)	core_model = nehalem (base cpu model which is used to mdoel haswell core)	core type=OOO
Fetch width	fetchWidth = 8 (fetch width in μ-ops per cycle)	NP	FETCH_WIDTH = 8 (max. μ-ops fetched per cycle)	NP	FETCH_BYTES_PER_CYCLE = 16 (ooo_core.cpp)
Fetch Buffer	fetchBufferSize = 16 (size in bytes)	FetchQueueSize = 16 (size of fetch queue in bytes)	ICACHE_FETCH_GRANULARITY = 16 (bytes of x86 code to fetch into decode buffer at once)	NP	same as fetch bytes/cycle
Fetch Queue (size set same as decode μ-ops queue size of Haswell pipeline)	fetchQueueSize = 56	UopQueueSize = 56 (Size of μ-op queue in num of μ-ops)	FETCH_QUEUE_SIZE = 56	NP	NP
No. of stages between fetch and decode	fetchToDecodeDelay = 3 (no. of stages = delay - 1)	NP	NP	NP	DECODE_STAGE (4) - FETCH_STAGE (1)
No. of stages between decode and rename	decodeToRenameDelay = 3 (no. of stages = delay - 1)	NP	NP	NP	NP
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
No. of stages between rename and issue	renameToIEWDelay = 4 (no. of stages = delay - 1)	NP	NP	NP	ISSUE_STAGE (11) - DECODE_STAGE (4)
No. of stages between issue and execute	issueToExecuteDelay = 2 (no. of stages = delay - 1)	NP	NP	NP	DISPATCH_STAGE (14) - ISSUE_STAGE (11)
No. of stages between writeback and commit	iewToCommitDelay = 4 (no. of stages = delay - 1)	NP	NP	NP	NP
Decode width	decodeWidth = 6 (Max. μ-ops decoded per cycle)	DecodeWidth = 4 (x86 instructions decoded per cycle)	FRONTEND_WIDTH = 6 (width in μ-ops per cycle of pipeline frontend)	NP	NP
Rename width	renameWidth = 6 (Max. μ-ops renamed per cycle using physical register file)	NP	FRONTEND_WIDTH = 6 (width in μ-ops per cycle of pipeline frontend)	NP	RF_READS_-PER_CYCLE 6
Dispatch width	dispatchWidth = 6 (Max. μ-ops that can be dispatched to instruction queue per cycle)	DisptachWidth = 6 (No. of μinstructions dispatched per cycle)	DISPATCH_WIDTH = 6 (no. of μ-ops dispatched to backend of pipeline per cycle)	dispatch_width = 6	ISSUES_PER_CYCLE 6
Issue width	issueWidth = 8 (Max. μ-ops that can be issued from instruction queue to functional units)	IssueWidth = 8 (No. of μinstructions issued per cycle)	MAX_ISSSUE-_WIDTH = 8	NP	determined by number of ports (8)
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
Squash width	squashWidth = 8 (Max. μ-ops squashed per cycle)	NP	NP	NP	NP
Writeback width	wbWidth = 8 (Max. width of writeback stage in μ-ops)	NP	WRITEBACK_WIDTH = 6 (in μ-ops)	NP	NP
Commit width	commitWidth = 8 (Max. μ-ops committed per cycle)	CommitWidth = 8 (No. of μinstructions committed per cycle)	COMMIT_WIDTH = 8 (No. of μ-ops committed per cycle)	commit_width = 4 (Max. instructions committed per cycle)	NP
Load queue entries	LQEntries = 72	LsqSize = 114 (Load store queue size in μ-ops)	LDQ_SIZE = 72	outstanding_loads = 72	ReorderBuffer< $72, 6 >$ loadQueue
Store queue entries	SQEntries = 42	LsqSize = 114 (Load store queue size in μ-ops)	STQ_SIZE = 42	outstanding_stores = 42	ReorderBuffer< $42, 6 >$ storeQueue
Instruction queue entries	numIQEntries = 60	IqSize = 60 (Instruction queue size in μ-ops)	ISSUE_QUEUE_SIZE = 16 (4 of them)	rs_entries = 60	WindowStructure< $1024, 60 >$ insWindow
No. of physical floating point registers	numPhysFloatRegs = 168	RfFpSize = 168	PHYS_REG_FILE_SIZE = 168	NP	NP
No. of physical integer registers	numPhysIntRegs = 168	RfIntSize = 168	PHYS_REG_FILE_SIZE = 168	NP	NP
Reorder buffer entries	numROBEntries = 192	RobSize = 192	ROB_SIZE = 192	window_size = 192	ReorderBuffer< $192, 6 >$ rob
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
Count of int Alu's	IntALU count = 4	IntAdd.Count = 4	ALU0,ALU1,ALU2,ALU3 (added new functional units by changing code)	NP	mod. in decode.cpp
Count of MultDiv units	IntMultDiv count = 2	IntMult.Count, IntDiv.Count = 2		NP	mod. in decode.cpp
Operation latency of int mul and div	IntMultDiv opLat = 4 and 1 cycle	IntMult.OpLat =4, IntDiv.OpLat = 20	OP_mulx=4, OP_divx = 20	mul = 4 , div =20 (static instruction costs)	mod. in decode.cpp
Count of floating point Alu's	FP_ALU count = 4	FloatSimple.Count, FloatAdd.Count = 4	FPU0, FPU1	NP	mod. in decode.cpp
Issue latency of floating point Alu operations	FP_ALU issueLat = 1 cycle	FloatSimple.IssueLat, FloatAdd.IssueLat = 1	OP_fx = 1 (x means all type of fp alu ops)	NP	mod. in decode.cpp
Operation latency of floating point Alu operations	FP_ALU opLat = 3 cycles	FloatSimple.OpLat, FloatAdd.OpLat = 3	OP_fx = 3 ()	fadd = 1 , fsub = 1 (static instructions cost)	mod. in decode.cpp
Count of floating point mul and div operations	FP_MultDiv count = 2	FloatMult.Count, FloatDiv.Count = 2		NP	mod. in decode.cpp
Issue latency of floating point mul and div operations	FP_MultDiv issueLat = 1 cycle	FloatMult.IssueLat, FloatDiv.IssueLat = 1	OP_fmul, OP_fdiv =1	NP	mod. in decode.cpp
Operation latency of floaitng point mul and div operations	FP_MultDiv opLat = 5, 15 cycles	FloatMult.OpLat, FloatDiv.OpLat = 5,15	OP_fmul, OP_fdiv = 5, 15	fmul = 5 , fdiv = 15 (static instruction costs)	mod. in decode.cpp
Cache Read Ports	ReadPort count = 2	NP	NP	NP	
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
Cache Write Ports	WritePort count = 1	NP	NP	NP	
Type of branch predictor	predType = Tournament	Kind = Combined	CombinedPredictor	Branch Predictor = Pentium_m	BranchPredictorPAg (2 level predictor)
Size of branch predictor tables	localPredictorSize = 2048 (size of local predictor) localCtrBits = 2 (bits per counter of local predictor) localHistoryTableSize = 2048 (size of local history table) globalPredictorSize = 4096 (size of global predictor) globalCtrBits = 2 (bits per counter of global predictor) choicePredictorSize = 4096 (size of choice predictor) choiceCtrBits = 2 (bits per counter of choice predictor)	Bimod.Size = 4096 (entires in bimodal predictor) Choice.Size = 4096 (entries in choice predictor) TwoLevel.L1Size = 1 (no. of branch history registers in branch history table of two level adaptive predictor) TwoLevel.L2Size = 1 (no of columns in pattern history table (second level table)) TwoLevel.HistorySize = 12 (size of branch history register)	METASIZE = 4096 (meta predictor size) BIMODSIZE = 4096 (bimodal predictor size) L1SIZE = 1 (level1 table size, size=1 makes this a g-share predictor) L2SIZE = 4096 (size of pattern history table of g-share) SHIFTWDITH = 12 (size of global history register of g-share)	Global Predictor = 4096 (hardcoded parameters in code) Bimodal Table = 4096 LoopBranchPredictor = 128 iBTB = 256	NB=12 (2pow(NB) entries for L1 Branch history shift registers) LB=12 (2pow(LB) entries for pattern history table (L2))
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
Branch taret buffer entries	BTBEntries = 4096	BTB.Sets = 1024, BTB.Assoc = 4	BTBSETS = 256, BTBWAYS = 4	NUM_ENTRIES = 4096 (associativity is set through NUM_WAYS = 4)	idealized
Return address stack size	RASSize = 16	RAS.Size = 16	RASSIZE = 16	NP	idealized
Branch misprediction penalty	14 (Implicitly set by configured delay between stages)	RecoverPenalty = 14 (No. of cycles fetch stage gets stalled after branch misprediction)	FRONTEND_STAGES = 14 (used to set branch misprediction penalty)	mispredict_penalty = 14 cycles	14 (Implicitly set by numbering of different pipeline stages)
L1 instruction cache access latency (cycles)	hit_latency + response_latency = 1+3 (hit_latency = hit latency of cache, response_latency = latency of sending response to core)	Latency = 4 cycles	NP	l1_icache data_access_time = 4	l1i latency = 4
L1 instruction cache associativity	assoc = 8	Assoc = 8	L1I_WAY_COUNT = 8	l1_icache associativity = 8	l1i ways = 8
L1 instruction cache miss status handling registers	mshrs = 16	MSHR = 16	NP	l1_icache_outstanding_misses = 16	l1i mshrs = 16
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
L1 data cache access latency	hit_latency + response_latency = 1+3 (hit_latency = hit latency of cache, response_latency = latency of sending response to core)	Latency = 4	LOADLAT = 4	l1_data_cache data_access_time = 4	l1d latency = 4
L1 data cache associativity	assoc = 8	Assoc = 8	L1_WAY_COUNT = 8	l1_data_cache associativity = 8	l1d ways = 8
L1 data cache miss status handling registers	mshrs = 16	MSHR = 16	MISSBUF_COUNT = 80 (sum of all MSHRs)	l1_data_cache outstanding_misses = 16	l1d mshrs = 16
L2 cache access latency	hit_latency + response_latency = 4+4 (hit_latency = hit latency from L1 to L2 cache, reponse latency = latency of sending response to upper level of cache)	Latency = 8	L2_LATENCY = 8	l2_data_cache access_time (11 cycles) + l1_data_cache tags_access_time (1 cycle)	l2 latency = 8
L2 cache associativity	assoc = 8	Assoc = 8	L2_WAY_COUNT = 8	l2_cache associativity = 8	l2 ways = 8
L2 cache miss status handling registers	mshrs = 32	MSHR = 32	MISSBUF_COUNT = 80 (sum of all MSHRs)	l2_cache_outstanding_misses = 32	l2 mshrs = 16
Note: **NP**=No parameter					

Parameter	Gem5 Parameter	Multi2Sim Parameter	PTLsim Parameter	Sniper Parameter	ZSim Parameter
L3 cache access latency	hit_latency + reponse_latency = 12+12 (hit_latency = hit latency from L2 to L3 cache, reponse latency = latency of sending response to upper level of cache)	Latency = 24	L3_LATENCY = 24	l3_cache data_access_time (31 cycles) + l2_cache_tags-_access_time (4 cycles)	l3 latency = 24
L3 cache associativity	assoc = 16	Assoc = 16	L3_WAY_COUNT = 16	l3_cache associativity = 16	l3 ways = 16
Cache line size	64	BlockSize = 64	CACHE_LINE_SIZE = 64	cache block_size = 64	lineSize = 64
Cache replacement policy	LRU	LRU	LRU	LRU	LRU
Main Memory latency	SimpleMemory latency = 57ns	Latency = 190 cycles	MAIN_MEM-_LATENCY = 190	dram latency = 57 (ns)	default in mem type DDR3-1333-CL10
Note: **NP**=No parameter					

Available pipeline configurations are also changed according to values in Table 3. The branch predictor modeled in Sniper is based on Intel Pentium M's branch predictor [43]. We have changed the table sizes of this branch predictor so that global predictor, bimodal predictor and BTB each are 4K entries in size. Pentium M branch predictor also contains a loop predictor (128 entries) and a direct-mapped iBTB (indirect branch target buffer) with 256 entries for indirect branches.

PTLsim is also quite flexible in terms of feature configurability. To set the pipeline and other basic configuration parameters some header files have to be changed. PTLsim directly fetches pre-decoded μ-ops from a basic block cache. Branch prediction is highly configurable in PTLsim. It includes a hybrid G-share and bimodal predictors and set-associative BTB, each set to 4k entries. PTLsim supports both distributed and shared issue queues. It is not possible to model a shared instruction queue as well as clusters in PTLsim. Thus, distributed instruction queues (with extra entries for compensation) are configured as done by Hayes et al [44].

Multi2Sim's configuration is set using two basic configuration scripts: one is used to set memory hierarchy configurations and the other is used to set pipeline configuration parameters. Both are configured according to target configurations. The configured branch predictor is a tournament predictor consisting of a bimodal predictor and two-level adaptive predictor in addition to a set associative BTB with similar sizes as other simulators.

ZSim needs changes in some headers files alongwith a configuration file to fully model a target architecture. We have changed the header files to add new functional units and configure different pipeline stages to model Haswell pipeline. ZSim does not support a tournament/ hybrid branch predictor. Therefore, a two level branch predictor (with table sizes as for other simulators' configured predictors) is modeled. Moreover, ZSim assumes an ideal branch target buffer.

As details of cache prefetchers on Haswell are not available, we did not configure any prefetcher on any simulator to eliminate one source of inaccuracy. We also deactivated cache prefetchers on real hardware before collecting reference performance statistics.

4.3 Experimental Workloads

We ran our experiments using SPEC CPU 2006 [45] and subset of the MiBench embedded benchmarks [46] benchmark suites.We simulated complete MiBench benchmarks, however; running complete SPEC benchmarks in simulated environment will take impractically long time. Thus, each application was executed for 500 million x86 instructions chosen from a statistically relevant portion of the program [47]. Simulation is fast-forwarded for all benchmarks and are warmed up for 100 million instructions before start of detailed simulation. We changed Multi2Sim and PTLsim's code to support the warmup functionality.

4.4 Performance Measurement on Real Hardware

We used PAPI [48] to measure IPC (instructions per cycle) values for entire execution of embedded benchmarks. In case of SPEC benchmarks we measured the IPC values for the same 500 million instructions interval that we simulated. Benchmarks were run five non-consecutive times and results are averaged to compare with simulation results and measure experimental error. We also used PAPI to measure number of cache misses and branch mispredictions on the real hardware. The compiler that we used is *gcc 4.4.7*. We started with 32-bit binaries for all simulators as multi2sim supports only 32 bit binaries, but many of them failed to work on gem5. So, we have used 64-bit binaries for gem5 only. The host OS used for compilation and

the execution of benchmarks is Scientific Linux 2.6.32. For gem5, experiments are performed on a Ubuntu 14.04 host OS. Reference hardware performance metrics are measured on both systems as well. We have used gem5's stable version of September 2015, Multi2Sim version 5.0, Sniper version 6.0 and PTLsim version available at [49] for all experiments.

5 Results and Analysis

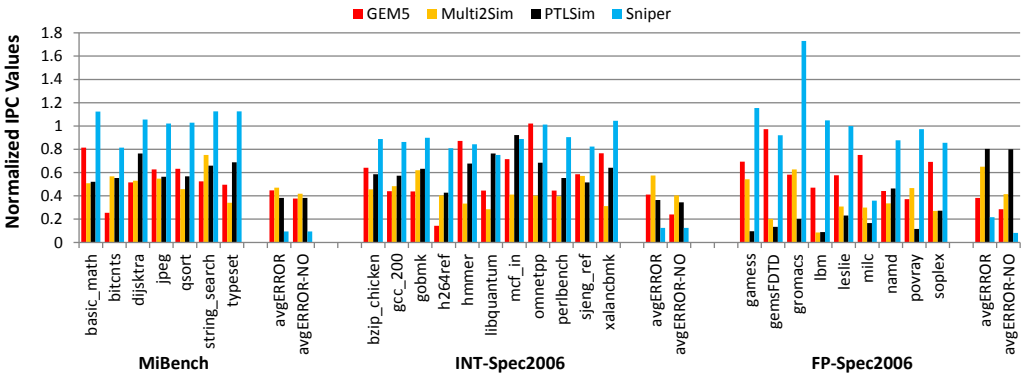

Figure 1: Normalized IPC values for all benchmarks

In this section, we compare the IPC, cache hit rate and branch misprediction results of the four simulators to that of real hardware runs on a Haswell processor. We also quantify the errors and evaluate the accuracy of this approach. All figures in this section show two types of average errors for all evaluated metrics: one including all benchmarks, $avgE$, and the other without outliers, $avgE-NO$; where an outlier corresponds to more than 50% inaccuracy in a metric. Figure 1 shows the normalized IPC values for all benchmarks to the real hardware results. From the figure we can see that Sniper shows the least error and only include one outlier. For embedded benchmarks, the mean absolute percentage error (MAPE) compared to real hardware runs are: 9.5%, 44.77%, 38.3% and 47.04% for Sniper, Gem5, PTLsim and Multi2Sim respectively. The MAPE values excluding outliers for Gem5 and Multi2Sim are 37.6% and 41.9% respectively.

The MAPE for integer benchmarks for Sniper, Gem5, PTLsim and Multi2Sim are 12.5%, 41.1% (24.0% excluding outliers), 36.5% and 57.4% (40.4% excluding outliers) respectively. For floating point benchmarks, the MAPE excluding outlier are 8.2%, 28.8% and 41.4% for Sniper, Gem5 and Multi2Sim respectively. PTLsim shows much greater inaccuracy in floating point benchmarks, with most of the benchmarks showing more than 50% error (PTLsim shows very high μ-ops to instructions ratio for most of them).

We have been able to observe various reasons which can contribute to overall inaccuracy shown by the simulators. For example, one of the primary reasons for the high underestimation of IPC for some benchmarks in PTLsim is the way x86 instructions are decoded into μ-ops. We observed that for all benchmarks including outliers, which have high inaccuracies in IPC values for PTLsim, the ratio of committed μ-ops to committed x86 instructions is very high compared to other simulators. For example, the ratio is 9 for *gemsFDTD* using PTLSim (on Gem5 and Multi2Sim, it is 1.36 and 2.27), 6.07 for *gamess* (on Gem5 and Multi2Sim it is 1.5 and 2.16), 5.43 for *povray* and 4.2 for *soplex*. As all pipeline widths are configured in number of μ-ops, thus if this ratio is very high, the overall IPC can diminish significantly.

Figures 2 and 3 show normalized L1 data cache misses and L3 cache misses per instruction for SPEC benchmarks. Figure 4 shows normalized values of branch mispredictions to hardware performance counters results. There are many cases where the average error goes above 100%. These figures help to understand very high underestimation of IPC values by simulators for some benchmarks. For example, most of the outliers in Gem5 (*h*264*ref*, *gcc*_200, *gobmk*, *perlbench*, *libquantum*, *namd*, *povray*) have much higher branch mispredictions and cache misses in comparison to real hardware. The reason for these inaccuracies in simulated branch predictors is that the modeled branch predictor fails to mimic the behavior of branch predictor of the real core for these benchmarks. The benchmarks that have high overestimated branch mispredictions have significant number of committed branch instructions (20% or more) compared to other benchmarks (4-5% committed branch instructions). This exposes the inefficiency of modeled branch predictor when compared to Haswell branch predictor. Some of the IPC inaccuracies can be due to the way some of the x86 instructions are decoded and implemented in Gem5. For example, integer division is implemented using loops of division μ-ops, where each μ-op computes a single bit of the quotient (each μ-op will have minimum one cycle latency). This implementation is different than the division algorithm on the real hardware. In addition, some inaccuracies in Gem5 were seen due to the mislabeled and inefficient microoperations [5].

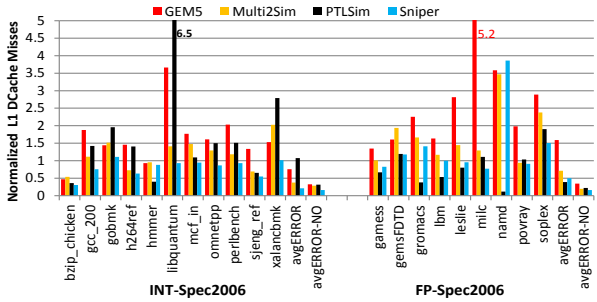

Figure 2: Normalized L1 DCache Misses

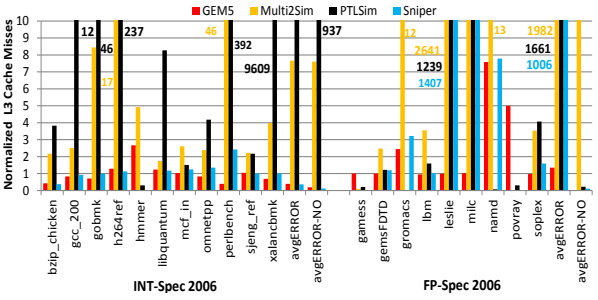

Figure 3: Normalized L3 Cache Misses

Similarly for Multi2Sim most of the outliers (*gcc*_200, *h*264*ref*, *hmmer*, *perlbench*, *gems*−*FDTD*, *namd*) have high overestimated branch mispredictions, that obviously highly affects overall IPC values. As observed for Gem5 as well, most of these benchmarks have high number of overall committed branch instructions (20% or more). It should be noted that the high overestimated branch mispredictions might not explain underestimated IPC values always. For example, branch mispredictions for *gromacs* on Sniper are highly overestimated, however; the total number of branch mispredictions is too low to affect overall IPC value (in fact Sniper highly overestimates the IPC value for *gromacs*, being the only outlier for Sniper). However, all examples mentioned for Gem5 and Multi2Sim previously have significant number of overall branch mispredictions that affects the overall IPC values. It is worth noting that most of the benchmarks that show highly overestimated L3 cache misses (*h*264*ref*, *xalancbmk*, *perlbench*, *leslie*3*d*, *namd*) by the simulators have low number of overall L3 cache misses on the real hardware. As a result this high inaccuracy might not affect the overall IPC accuracy significantly. Another reason for high L3 cache misses inaccuracy can be accumulation of inaccuracies from lower level of caches into L3 cache. We also noted that the inaccuracy in L2-$ misses to L3-$ misses ratio is lower than the inaccuracy of L3-$ misses alone.

Some of the inaccuracies can be a result of lack of support of fused μ-ops, and μ-op cache of Haswell (which can significantly reduce the effective pipeline depth in case of μ-op cache

hits) by the simulators. The abstract level of some simulators and the lack of other micro-architectural details can also induce errors in simulation models.

To observe the effect of changes in simulator configurations on simulated performance, or relative performance, we ran the simulations for embedded benchmarks after halving pipeline stages' width from Table 3. Figure 5 shows the IPC values of those runs normalized to IPC values with normal pipeline width. It is not easy to say what should be the exact effect of such a change for each benchmark, however; the figure shows that Sniper seems to have the largest difference in relative performance and thus the most sensitive to the width of stages.

Figure 5 and 6 show the percent change in IPC values for pipeline width and cache size change respectively, from the normal configurations in table III. For many benchmarks, the simulators show similar relative change in performance. PTLsim is the least sensitive on average among all other simulators. Multi2Sim seems to be more sensitive to change in cache sizes than other simulators as its memory mode is pretty detailed. Sniper and gem5 show similar sensitivity to cache size change. They also show, on average, very close IPC change in pipeline width, being more sensitive than PTLSim and Multi2Sim.

In addition to comparing the simulated results of the simulators, we compare the time it takes for each simulator to simulate 500 million instructions including the fast forwarding time. Figure 7 depicts the average simulation time for each simulator for the different types of benchmarks. As shown from the figure, Sniper is the fastest simulator, followed by PTLsim. gem5 and Multi2Sim show close simulation time on average. It should be noted that the average simulation time is based on only a subset of overall simulation experiments, which were run in an isolated environment on host systems.

Our main observations based on our experiments are following:

- The main sources of inaccuracies in simulated statistics are highly overestimated branch mispredictions, imprecise decoding of instructions into microoperations, high cache misses and lack of modeling of all optimization structures of real hardware.

- Although it is widely accepted that simulators cannot provide speed and accuracy simultaneously, for the given target, Sniper shows the most accurate and fastest results out of those studied simulators. Sniper is fast because its simulation model combines both interval and cycle-level simulation. Moreover, Sniper shows the least experimental error as it is the only one of the four simulators that has been modified and validated for x86 architectures, including Nehalem μ-architecture [32]. On the other hand, we did not find any validation efforts for x86 processors for other simulators.

- An uncalibrated/unvalidated simulator for a particular architecture can show significant simulated performance differences when compared to real hardware. This is also demonstrated in an attempt to calibrate MARSSx86 for a particular target machine [**?**].

- A more accurate simulator may still not be able to fit your needs. For instance, Sniper though shows greater accuracy, is not very flexible to allow one to model new micro-architectural features compared to Gem5. On the other hand, Gem5 and PTLsim are more flexible and can help in studying performance of particular micro-architectural blocks.

- For given workloads Sniper appears to be the most accurate, for other type of workloads specially full system workloads it might not show similar accuracy as it does not have extensive device modeling support. On the other hand Gem5 might be better suited for such workloads as it supports full system simulation and supports vast device models.

- Better accuracy and high speed makes Sniper a convincing choice for many-core x86 architectures (specially modern Intel processors like). Gem5's detailed processor modeling, multi-ISA support, full system simulation support and active development community makes it a good choice to perform detailed experiments on a particular processor sub-system, to study OS interaction with hardware or to study interaction of an x86 core with a different ISA core. Multi2Sim can be good option for CPU-GPU architecture simulation. PTLsim can serve as base x86 simulator to develop more diverse simulators as done in the case of MARSSx86 [24] simulator.

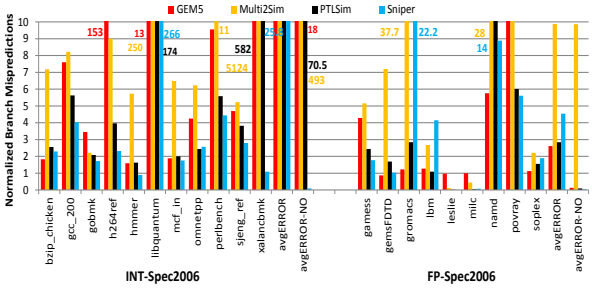

Figure 4: Normalized branch mispredictions

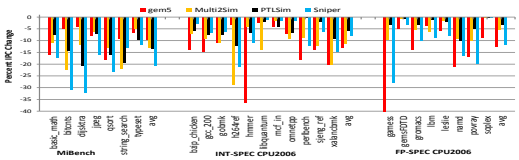

Figure 5: IPC change for half pipeline width

6 Conclusion

In this paper, we presented a comprehensive study of x86 architectural simulators. We have surveyed and compared many x86 computer architecture simulators and grouped them in respective categories. We performed verification tests of four modern computer architecture simulators and measured their experimental error compared to real hardware runs. The experimental error rate shown by the simulators does not necessarily mean that main cause are bugs in the simulator. Some reasons of inaccuracies include: not modeling all microarchitecture optimization details as real hardware, which makes it harder to validate; varying degree of flexibility and configurability; inaccurate decoding of instructions into microoperations; and different labeling of microoperations of simulators. The results show that Sniper has the least absolute error. In terms of single-core simulations speed, Sniper comes out at the top with

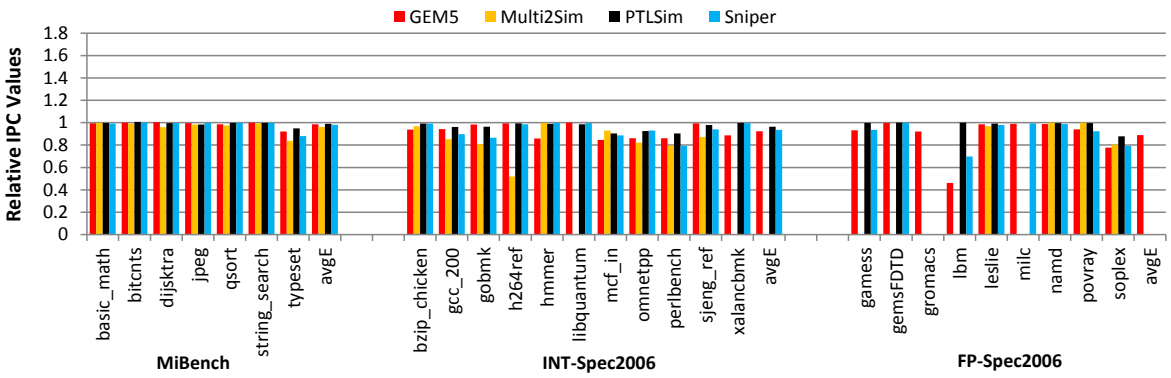

Figure 6: IPC change for reduced cache size

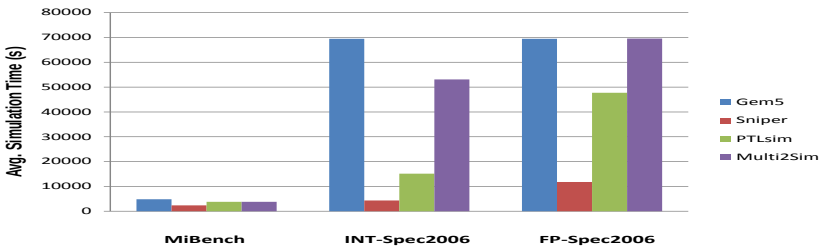

Figure 7: Average simulation time

shortest simulation time. However, choosing a simulator can differ depending on the main focus of the research. For example, Sniper is targeted for multicore simulations and is the most accurate among the studied simulators, however; it does not produce detailed performance statistics for simulated system and not very flexible compared to the others. It can be a best choice for some symmetric and asymmetric x86 multicore research project but not for multi-ISA heterogeneous multicore because it only supports x86. On the other hand, gem5 produces very detailed results and can be helpful in studying only particular blocks of processor and supports multiple ISAs. In future work, we will consider digging deeper to figure more sources of inaccuracies by performing further experimentation using certain μ-benchmarks to find more sources of inaccuracies. We will also study some other new x86 simulators.

References

[1] K. Skadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai, "Challenges in Computer Architecture Evaluation," *Computer*, vol. 36, pp. 30–36, August 2003.

[2] `http://www.diva-portal.se/smash/get/diva2:829764/FULLTEXT01.pdf`.

[3] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 449–458, 2009.

[4] R. A. Uhlig and T. N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, pp. 128–170, 1997.

[5] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, pp. 4–12, Dec. 2015.

[6] R. D. D. B. S. Keckler, "Measuring experimental error in microprocessor simulation," in *ISCA*, 2001.

[7] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *ACM Int. Conf. for High Performance Comp., Net., Storage and Analysis*, pp. 1 – 12, 2011.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 Simulator," *SIGARCH Comp. Arch. News*, vol. 39, pp. 1–7, May 2011.

[9] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2sim: A simulation framework to evaluate multicore-multithreaded processors," in *Int. Symp. on Comp. Arch. and High Perf. Comp.*, pp. 62–68, Oct. 2007.

[10] M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *IEEE Int. Symp. on Perf. Analysis of Systems & Software*, pp. 23–34, 25-27 April, 2007.

[11] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer*, vol. 35, p. 59.

[12] P. S. Magnusson, F. Larsson, A. Moestedt, B. Werner, J. Nilsson, P. Stenström, F. Lundholm, M. Karlsson, F. Dahlgren, and H. Grahn, "SimICS/Sun4m: A VIRTUAL WORKSTATION," in *Usenix Annual Technical Conference*, pp. 119–130, Jun. 1998.

[13] T. Sherwood and J. Y. Joshua, "Computer Architecture Simulation and Modeling," *IEEE Micro*, vol. 26, pp. 5–7, July/August 2006.

[14] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset," *SIGARCH Comp. Arch. News*, vol. 33, pp. 92–99, November 2005.

[15] `http://parsa.epfl.ch/simflex/`.

[16] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *IEEE ISPASS*, pp. 74–85, April 2013.

[17] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "Simflex: A Fast, Accurate, Flexible Full-System Simulation Framework for Performance Evaluation of Server Architecture," *ACM SIGMETRICS Perf. Eval. Review*, vol. 31, pp. 31–34, Mar. 2004.

[18] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," in *ISCA*, pp. 84–95, 9-11 June 2003.

[19] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval Simulation: Raising the Level of Abstraction in Architectural Simulation," in *IEEE Int. Symp. on High Perf. Comp. Arch.*, pp. 1–12, 9-14 Jan. 2010.

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

[21] G. H. Loh, S. Subramaniam, and Y. Xie, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *IEEE ISPASS*, pp. 53–64, April 2009.

[22] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *Int. Symp. on Comp. Arch.*, vol. 41, pp. 475–486, June 2013.

[23] L. Schaelicke and M. Parker, "Ml-rsim reference manual," *Dept. of CSE, Univ. of Notre Dame, Tech. Rep. TR*, pp. 02–10, 2002.

[24] A. Patel, F. Afram, and K. Ghose, "MARSS-x86: A Qemu-Based Micro-Architectural and Systems Simulator for x86 Multicore Processors," in *Int. QEMU Users Forum, held in conjunction with Design Automation and Test in Europe*, pp. 29–30, 18 Mar. 2011.

[25] P. Crowley and J.-L. Baer, "On the Use of Trace Sampling for Architectural Studies of Desktop Applications," in *Workload Characterization: Methodology and Case Studies*, pp. 15–24, Dallas, TX, 1999.

[26] A. Sharma, A.-T. Nguyen, J. Torellas, M. Michael, and J. Carbajal, "Augmint: A Multiprocessor Simulation Environment for Intel x86 Architectures," Tech. Rep. 1463, Center for Supercomputing Research and Development, UIUC, 28 Mar. 1996.

[27] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *Int. Conf. on Parallel Arch. and Compilation Techniques*, pp. 335–344, 2012.

[28] 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*, vol. 26, pp. 52–60, July/August 2006.

[29] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of Error in Full-System Simulation," in *ISPASS*, pp. 13–22, 2014.

[30] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy Evaluation of GEM5 Simulator System," in *Int. Workshop on Reconfigurable Communication-centric Systems-on-Chip*, pp. 1–7, 2012.

[31] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A Distributed Parallel Simulator for Multicores," in *IEEE Int. Symp. on High Perf. Comp. Arch.*, pp. 1–12, Jan. 2010.

[32] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM TACO*, vol. 11, no. 3, p. 28, 2014.

[33] `http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/`.

[34] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, "PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education," in *Proceedings of the workshop on Comp.arch. education: held in conjunction with ISCA*, p. 22, June 2004.

[35] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *IEEE/ACM Int. Symp. on Microarch.*, pp. 469–480, 12-16 Dec. 2009.

[36] `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`.

[37] http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html.

[38] http://www.agner.org/optimize/instruction_tables.pdf.

[39] http://www.realworldtech.com/haswell-cpu/.

[40] http://www.anandtech.com/show/6355/intels-haswell-architecture/6.

[41] http://xania.org/201602/haswell-and-ivy-btb.

[42] R. E. Kessler, E. J. McLellan, and D. A. Webb, "The Alpha 21264 Microprocessor Architecture," in *ICCD: VLSI in Computers and Processors*, pp. 90–95, 1998.

[43] V. Uzelac and A. Milenković, "Experiment Flows and Microbenchmarks for Reverse Engineering of Branch Predictor Structures," in *IEEE ISPASS*, pp. 207–217, 2009.

[44] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero, "Vector extensions for decision support dbms acceleration," in *IEEE/ACM MICRO*, pp. 166–176, 2012.

[45] "SPEC CPU 2006." https://www.spec.org/cpu2006/.

[46] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE WW*, pp. 3–14, Dec. 2001.

[47] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *the 10th Inter. Conf. on Architectural Support for Progr. Languages and Oper. Sys.*, pp. 45–57, October 2002.

[48] "Performance Application Programming Interface." http://icl.cs.utk.edu/papi/. [Online; accessed 5-August-2015].

[49] https://github.com/stephand/ptlsim.