IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 30, 2019, accepted May 3, 2019, date of publication May 20, 2019, date of current version June 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2917698

A Survey of Computer Architecture Simulation Techniques and Tools

AYAZ AKRAM[1] AND LINA SAWALHA[2], (Senior Member, IEEE)
[1]Department of Computer Science, University of California at Davis, Davis, CA 95616, USA
[2]Department of Electrical and Computer Engineering, Western Michigan University, Kalamazoo, MI 49008, USA

Corresponding author: Lina Sawalha (lina.sawalha@wmich.edu)

This work was supported in part by grants from the Faculty Research and Creative Activities Award (FRACAA) and the College of Engineering and Applied Sciences at Western Michigan University.

ABSTRACT Computer architecture simulators play an important role in advancing computer architecture research. With wider research directions and the increased number of simulators that have been developed, it becomes harder to choose a particular simulator to use. This paper reviews the fundamentals of different computer architecture simulation techniques. It also surveys many computer architecture simulators and classifies them into different groups based on their simulation models. Comparing computer architecture simulators with each other and validating their accuracy have been demanding tasks for architects. In addition to providing a survey of computer architecture simulation tools, we measured the experimental error of six contemporary computer architecture simulators: gem5, MARSSx86, Multi2Sim, PTLsim, Sniper, and ZSim. We also performed a detailed comparison of these simulators based on other features such as flexibility and micro-architectural details. We believe that this paper will be a very useful resource for the computer architecture community especially for early-stage computer architecture and systems researchers to gain exposure to the existing architecture simulation options.

INDEX TERMS Computer architecture simulators, simulation techniques, validation, x86 simulators, simulators evaluation.

I. INTRODUCTION

Computer architects use simulation to assess different design options, test new research ideas and analyze the performance/power consumption of different processor models. Analytical models are not suitable for evaluating architectural/microarchitectural designs and design variations as they produce inaccurate results because of the huge amount of configurations and small details that can cause small variations in performance. Simulation is considered to be the standard performance modeling method [1]. The majority of published research is based on the use of simulators to analyze the performance of new ideas. Many computer architecture simulators support various instruction set architectures (ISAs), microarchitectures and are based on different simulation models ranging from trace-based to cycle-accurate. It can be a daunting task for new researchers in the area of computer architecture and systems, to choose one simulator and start their research. In addition, in order to have a trust in simulation studies, simulation results need to be validated. This can be challenging, especially when there is not enough documentation about simulators. There is little work that evaluates current computer architecture simulators, performs a comparison among them, and/or compares their accuracy to contemporary processors. Our major contributions in this paper are:

- Providing an up-to-date survey of computer architecture simulation techniques and simulators.
- Categorizing, analyzing and comparing various computer architecture simulators, which can help the community to understand the use-cases of different simulation tools.
- Providing detailed characteristics and experimental error comparison of six modern *x86* computer architecture simulators: gem5 [2], Multi2sim [3], MARSSx86 [4], PTLsim [5], Sniper [6], and ZSim [7].
- Reviewing the most important challenges for architecture simulators and the solutions that have been proposed to resolve those issues.

The associate editor coordinating the review of this manuscript and approving it for publication was Stavros Souravlas.

[1]A. Akram finished the work while at Western Michigan University.

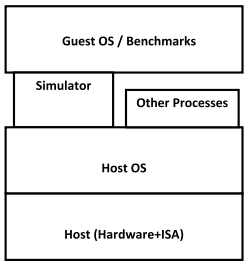

FIGURE 1. Simulator running on a host machine.

In computer architecture, the main goal of simulation is to model new research ideas for parts of a computer system (e.g. microprocessor, memory, IO devices) or a complete computer system and estimate the performance improvements and/or power consumption. Simulators also help computer architects in evaluating, debugging and understanding the behavior of existing systems. In simulation terminology, a computer system that is being simulated is called a *target*, and the system where the simulation is run is called a *host*. Workloads that run on simulators can be standard test programs, known as benchmarks, which are run to assess the performance of a processor or a computer. The workload being simulated can also be an operating system (OS), sometimes referred as a *guest OS*. The interaction between a simulator and a host system is shown in Figure 1. This survey is a more comprehensive and an updated survey compared to previous existing surveys, which either focused on teaching related simulators [8]–[11], memory simulators [12], [13] or were not much detailed. Nikolic *et al.* [10] have surveyed many computer architecture simulators suitable for teaching computer architecture courses. They evaluated various simulators based on the criteria of topics covered in the classroom, and simulation features. Uhlig and Mudge [12] and Holliday [13] discussed different memory simulation techniques, which use reference address traces. Urden [14] compared and evaluated the performance results of three different computer architecture simulators against each other. However, his study did not compare the simulators with real hardware runs. In this survey, we compare the simulators' results with that of real hardware experiments to measure their inaccuracies. Nowatzki *et al.* [15] discussed various pitfalls associated with the usage of architectural simulators. They have also discussed the errors they observed in four performance and power simulators: gem5 [2], GPGPUSim [16], McPAT [17] and GPUWattch [18]. Validation efforts for various simulators (e.g. SimpleScalar, SMARTS, Microlib, gem5, Sniper, SiNUCA, Ramulator) have also been published [19]–[25]. These papers focus only on one simulator that is being validated and usually do not include comparisons with related tools. This survey is up-to-date, which includes newer processor architecture simulators, and it is more detailed compared to previous surveys. The survey also compares and contrasts several modern simulators. In order to accelerate the process of simulation, researchers often rely on sampling techniques. The paper also discusses few of the commonly used sampling methods associated with simulating computer architecture components. Similarly, the accuracy of simulation results is always a concern for architects. We briefly explore different existing simulator validation approaches. Furthermore, this work compares the experimental error of six x86 simulators with hardware runs and provides the relative performance of the simulators when changing some microarchitectural configurations. Finally, we pinpoint some causes of inaccuracies in the simulation results that we observed.

Scope of the paper: Because computer architecture research covers a wide range of architectures, from processor microarchitectures to special purpose architectures and accelerators, which use different types of simulators, we limit the scope of the paper to review in details computer architectural and microarchitectural simulators and simulation models of processors. The paper mentions some uncore and accelerator simulators. It does not survey in details *standalone* specialized simulators for mircorachictectrual structures or uncore components; however, it does mention detailed microarchitectural or uncore simulations as features of those full computer architectural/microarchitectural simulators. It also discusses those full simulators that can also simulate accelerators or can be attached to accelerator simulators. The paper does not cover systems on chip (SoC) simulators, although many of the simulators that are discussed are capable of simulating embedded processors. The paper also discusses different simulation methodologies, categorizes them, and compares and contrast those techniques. In addition, the paper discusses the challenges associated with simulation and their possible solutions, and discusses simulation evaluation techniques. However, to limit the scope, the paper does not cover in details specific implementation limitations, such as handling target multi-threading.

The organization of the rest of the paper is as follows: Section II classifies simulators into different categories and discusses these categories in details. Section IV summarizes the different existing computer architectural/microarchitectural simulators. Section V explores the challenges faced in computer architecture simulation and their solutions. Section VI briefly discusses the validation of simulators. Section VII describes in details six modern x86 simulators that we have chosen for detailed evaluation. Section VIII discusses the methodology used to measure the experimental error of simulators and their relative performance. Section IX shows the evaluation results of the x86 simulators. Finally, we conclude the paper in section X.

II. CLASSIFICATION OF SIMULATORS

Simulators can be classified into various groups on the basis of three most important factors: detail of simulation, scope of the target and input to the simulator. This section discusses in details the aforementioned classification taxonomy. It should be noted that this classification is not mutually exclusive and one simulator can belong to more than one class. In addition,

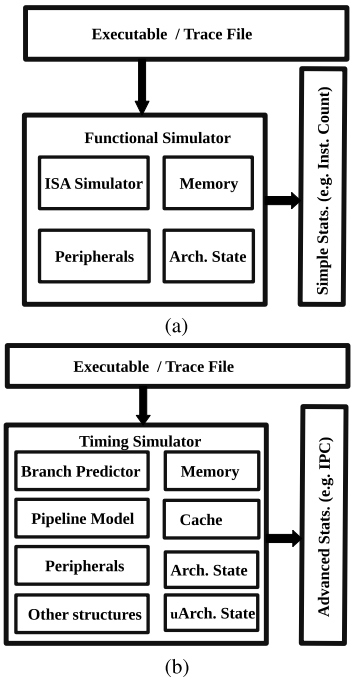

FIGURE 2. Simulators based on simulation details (adapted from [26], p. 492, Figure 9.2). (a) Functional Simulator. (b) Timing Simulator.

some simulators are classified based on certain aspects or specializations, which is also discussed in this section.

A. CLASSIFYING SIMULATORS BASED ON THE DETAIL OF SIMULATION

An important factor to classify simulators is the level of detail that any simulator implements in its design. The main classes of simulators based on simulation detail are functional, timing and functional/timing simulators.

1) FUNCTIONAL SIMULATORS

A functional simulator implements the architecture only and focuses on achieving the same functionality of the modeled architecture. In other words, functional simulators behave like emulators (emulate the behavior of target's instruction set architecture (ISA)). They are usually faster than the other types of simulators, but they cannot keep track of detailed microarchitectural parameters, as a program runs on the simulator, because they do not implement the microarchitecture. While developing new instruction sets, functional simulators can be used for testing purposes. Moreover, functional simulators can help in identifying architectural features of a program's execution, for instance, the total number of different types of instructions in a program, memory access locality, etc. Figure 2(a) shows a block diagram of a functional simulator.

SimpleScalar simulator [27] has been used for teaching and research purposes. SimpleScalar is a comprehensive toolset. It has various simulation models, out of which *sim-safe* is an example of a functional simulation model. It is a minimal SimpleScalar simulator that only simulates the ISA. A speed-optimized version of *sim-safe* is named as *sim-fast* [27]. Simics [28] is another functional simulator, which has a unique ability of executing a program in forward or backward directions. SimCore [29] is a functional simulator for Alpha processors. It is claimed to be 19% faster than *sim-fast* of SimpleScalar toolset [29]. EduMIPS64 [30], a visual functional simulator written in java for MIPS, was designed to be used in classrooms for teaching computer architecture courses. HASE [31] is a tool for high-level simulation and visualization of computer architectures. It was developed in the 90's using object oriented simulation languages. HASE project provides many computer architecture models targetting teaching concepts related to computer architecture. Barra [32] is a functional simulator for GPGPU (general purpose graphics processing units). It supports simulation of CUDA applications. Another example of functional simulators is the 'AtomicSimple' CPU model of *gem5*. One alternative to creating a functional simulator is to instrument a program's binary with a code that is responsible for collecting the required information when the program executes on a real hardware [33]. Such tools are called dynamic binary instrumentation tools, for example, Pin tools [34]. There are many simulators (e.g. CMP$im [35], Sniper [6]), which also rely on instrumentation tools to perform functional simulation.

2) TIMING SIMULATORS

Timing simulators, also known as performance simulators, simulate the microarchitecture of processors (Figure 2(b)). They produce detailed statistics about the timing/performance of a target system [26]. For instance, in case of the simulation of a processor, this information might comprise statistics like instructions per cycle (IPC), program run time, performance of a memory system and other detailed microarchitecture-related statistics. It is not required for a timing/performance simulator to emulate the functionality of a target. Timing simulators have different subtypes, depending on the degree of details included in the simulator: cycle-level simulators, event-driven simulators and interval simulators.

Cycle-level Simulators: Cycle-level simulators simulate an architecture by imitating the operation of the simulated processor for each cycle. In contrast to cycle-accurate simulators that simulate *accurately* what happens on each cycle using RTL implementation [36], cycle-level simulators do not model the hardware with minute details. Cycle-level simulators are slow and utilize a considerable amount of memory compared to functional and other timing/performance simulators.

For instance, *sim-fast* (the fastest functional simulator for SimpleScalar) can simulate instructions 25 times faster than the detailed cycle-level simulation model of SimpleScalar. The cycle-level performance model of SimpleScalar is called *sim-outorder*, which is a detailed microarchitectural timing model. It implements an out-of-order superscalar processor that supports speculation. Most of the design parameters are

configurable by users, for example, the number and latency of functional units, instruction queue and reorder window sizes, memory latency, etc. Another example of cycle-level simulators is MSim [37], which is a multi-threaded microarchitectural simulation environment for Alpha processors that simulates major pipeline components. MSim is based on SimpleScalar.

Event-driven Simulators: An event-driven simulator simulates a target based on events instead of cycles. Usually, they make use of event queues. Simulation jumps to the time when an event is scheduled, based on the event queues, instead of going through all cycles. That way, simulators can save time by not simulating the cycles for which there are no scheduled events [38]. Often, some parts of a simulator are modeled on a cycle-level, while others are event-driven. For example, the work done by Reilly and Edmondson [39] to simulate the performance of Alpha microprocessors. An important point to note here is that often literature does not distinguish between cycle-level and event-driven simulators.

SESC [40], a relatively fast simulator is an example of an event-driven timing simulator that supports MIPS ISA. SESC supports various simulation models such as single processors, chip multiprocessors (CMPs), processor in memory (PIM). RSim [42] is another example of event-driven timing/performance simulators. It was developed in the 1990's and focused on both instruction-level parallelism (ILP) and shared memory multiprocessors. Detailed accuracy in RSim is achieved at the cost of a slow speed [42]. Sampling microarchitecture simulation (SMARTS) [20] framework and Flexus (Simics) [43] form the basis of a cycle-level timing simulator SimFlex [44]. Some of the components of this simulator are event-driven internally. SimFlex can perform fast simulation of uniprocessor and multiprocessor systems. It supports various memory models, but implements a simple in-order CPU model.

Interval Simulators: With the diversion of research focus towards multi-core and many-core systems, researchers have been looking for new simulation techniques that balance simulation accuracy and speed as alternatives for cycle-level and event-driven only simulators. For instance, interval simulation [45] is one of such recently proposed techniques. This technique makes use of the fact that regular instruction flow through the pipeline can be broken down into sets of intervals based on miss events (cache misses, branch mispredictions). Special purpose portions of architectural simulators, like branch predictors and memory system, can be used to simulate the miss events and find their exact timings. Then, these timings along with an analytical model are used to estimate the duration for every interval of instructions.

3) INTEGRATED TIMING AND FUNCTIONAL SIMULATORS

Functional simulators are often integrated with timing simulators to achieve a more flexible and accurate simulation model. The two types of simulators might or might not be coupled together. The technique of coupling the simulators, in which instructions execute at the execute stage of the modeled pipeline, is known as *execute-in-execute*. This makes *execute-in-execute* a relatively complicated technique as compared to decoupling. On the other hand, it can increase the accuracy of the modeled timing-dependent instructions like synchronization and IO operations [46]. gem5 [2] is an example of a simulator that uses this technique. To simplify the development and reduce its complexity, often simulators decouple functional and timing (performance) simulation. Usually, some third party software is used for functional simulation. For example, Simics [28] is used by both SimFlex [44] and GEMS [38] for functional simulation, and Pin [34] is used by both Graphite [47] and Sniper [6] (based on Graphite) for functional simulation. For decoupling, there are three practical types [33]: timing-directed, functional-first and timing-first. In case of a timing-directed simulator, the timing model leads the simulation and gives directions to a functional model to execute instructions. This makes it possible for timing directed simulators to model speculative paths. Functional-first simulators use functional models to generate instruction traces which are fed to a timing model to derive detailed simulation. Since the functional models only executed instructions on the correct-path, it is hard to model speculative paths with this kind of simulators. In the case of timing-first simulators, the timing-simulator also executes instructions and uses a separate functional model to verify its execution. Figures 3(a), 3(b) and 3(c) illustrate these three types.

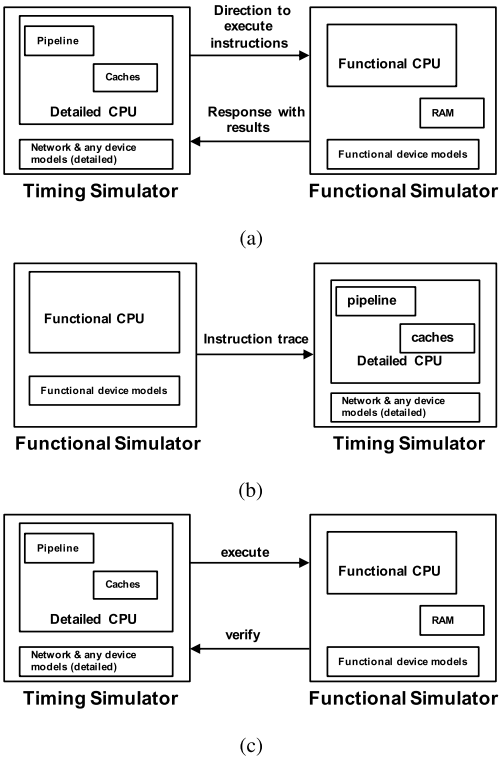

FIGURE 3. Three types of integrated timing and functional simulators. (a) Timing-directed Simulator. (b) Functional-first Simulator. (c) Timing-first Simulator.

Timing-directed Simulators: In this category, a functional simulator records the architectural state (e.g. register and memory values) of the processor being simulated. The timing simulator, which has no idea of data values on its own, takes and uses these values from the functional simulator to perform a specific task when required [33]. The functional model and the timing models interact heavily in this type of simulators as the timing model directs the functional model and the functional model feeds values to the timing model. This interaction makes this simulation model suitable for modeling architectures with dynamically changing functional behavior, such as multicore architectures [48]. For example, for a load instruction the functional model computes the instruction's effective address, and the timing model uses this address to determine if the load is causing a cache miss. The returned value from the cache or the memory, will eventually be read by the functional simulator. Asim [49] is an example of this category of simulators.

Functional-First Simulators: In this simulation model, the functional simulator runs prior to the timing simulator and generates an instruction trace (a stream of instructions) that feeds the timing simulator at runtime. In the case of conditional branches, the functional simulator always follows the correct path and it cannot simulate the behavior of branch predictors [50]. If there is a mispredicted branch in the timing simulator's pipeline, the functional simulator restores its previous state before the branch and continues along the mispredicted path. Later, the pipeline has to be flushed due to this mispredicted branch. Since the timing simulator always lags behind the functional simulator, there can be ordering problems while simulating more than one thread [50]. For instance, the time at which the functional model reads a memory value in case of a *load* instruction can be different from the time when the timing model requests the same value, and this can result in reading different values. This problem can be resolved by a speculative functional-first simulation [33]. In this technique, whenever a timing model detects that the data it reads is different from the data that the functional model has read, it asks the functional model to restore the processor's state to the state before the load instruction and then it executes the load instruction with the correct data. As, timing and functional models run in parallel, there is an opportunity to exploit this parallelism for better performance of the simulator. This type of simulators has much better performance as compared to timing-directed simulators, because it is not required for the timing model to direct the functional model at every instruction or cycle as in timing directed simulators. SimWattch [51] is an example of functional first simulators. SimWattch integrates Simics with Wattch [52]. Wattch is based on SimpleScalar and simulates both power and performance.

Timing-First Simulators: In this approach, timing simulators run ahead of functional simulators [50]. Timing simulators simulate the microarchitecture of a target processor at the cycle-level. Timing simulators usually use functional simulators for verification of functional execution of all instructions. The instruction is retired in case of a match between the architectural state of both the functional and the timing simulators. In case of a mismatch, the timing simulator recovers by flushing the pipeline and restarting the instruction fetch following the problematic instruction. As such, the timing simulator makes forward progress. If these recoveries happen frequently, they can impact the simulated system's timing, and thus accuracy, depending on the depth of the simulated pipeline [50]. GEMS [38], FeS_2 [53] and Multi2sim [3] are some examples of timing first simulators.

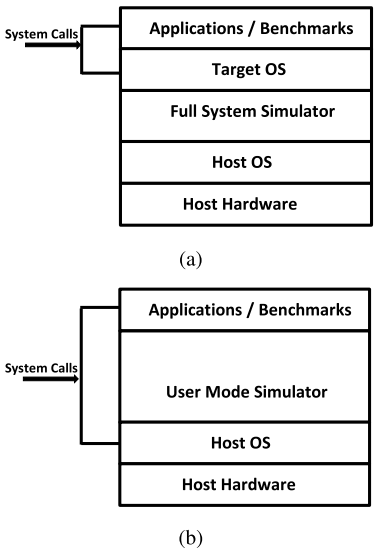

FIGURE 4. Simulators based on scope of target (adapted from [26], p. 491, Figure 9.1). (a) Full System Simulator. (b) User Mode Simulator.

B. CLASSIFYING SIMULATORS BASED ON THE SCOPE OF THE TARGET

Another factor to consider while categorizing simulators is the scope of the target system that is being simulated. Simulators can be classified into two types based on scope:

1) FULL-SYSTEM SIMULATOR

Given any supported operating system (OS) binary, a full system simulator is able to completely boot that OS and run application benchmarks on that OS as they would run normally on a real target machine. Figure 4(a) shows the basic functionality of a full system simulator. A full-system simulator simulates all needed I/O devices, memory and network connections that are required to boot and run any system. Applications that run on a simulated target system execute their system calls directly on the target system [26]. As a result, this form of simulation is complicated and time consuming. Ideally the OS should not be modified, but sometimes binaries of the OS are customized to make the process of OS booting less resource consuming.

Full-system simulators may or may not be timing simulators. For instance, gem5 [2] is an example of a full-system timing simulator, while SimOS [54] is a full-system

functional simulator. gem5 has the ability to boot Linux, Solaris and Android operating systems on specific hardware that it supports. SimOS developed in the late 90's, was one of the first full-system simulators (not currently active). Sunflower [55] suite includes a full-system simulator with microarchitecture and IO modeling. Simics is able to boot unmodified OS. SimFlex [44], ML-RSim [56], MARSSx86 [4] and PTLsim [5] are other examples of this type of simulators.

2) APPLICATION LEVEL/USER MODE SIMULATOR

These simulators run only target applications instead of simulating a complete OS. They simulate microprocessor and limited peripherals. In this kind of simulators, system calls are usually bypassed by the simulator and are serviced by the underlying host operating system, as shown in Figure 4(b). It might not be a problem to simulate only user-mode code for benchmarks that execute system-level code for a short duration (e.g. compute intensive benchmarks as SPEC CPU2006 and CPU2017) [33]. However, for benchmarks that spend a significant time to execute system-mode code, user mode simulation is not enough–(e.g. server related benchmarks such as Web-Bench and NetBench, real world performance benchmarks for example SYSMARK, and transaction processing and database benchmarks such as TPC-C). For multithreaded workloads, OS scheduling should be taken into account as it affects workload performance. Thus, it is necessary to simulate OS-level effects to get a better estimation of performance. On the other hand, application-level simulators are usually less complex and fast as compared to full system simulators. SimpleScalar is the most known example of application-level simulators. SESC [40], Sniper [6], and RSim [42] are other examples of application only simulators.

C. CLASSIFYING SIMULATORS BASED ON THE INPUT TO THE SIMULATOR

We can categorize simulators into two categories based on the input to the simulator itself, traces or executables.

1) TRACE-DRIVEN SIMULATORS

Trace files are used as inputs to trace-driven simulators. These trace files are prerecorded streams of instructions executed by benchmarks with some fixed inputs. As benchmarks execute on real machines statistics including instruction opcodes, data addresses, branch target addresses, etc are recorded in a trace file. Trace-driven model makes the implementation of the simulator simple. Trace-driven simulators can be easily debugged because experimental results can be reproduced. The size of trace files can be huge, which poses limits on the total instruction count in each trace file and/or the number of trace files used at once, and may lead to a slower simulation time [33], [57]. Different trace sampling and trace reduction techniques [58], are used to resolve the problem of large size of trace files. Apart from this, these simulators usually do not model execution of mispeculated code, which can affect performance estimation results of structures such as branch predictors. To solve the problem of branch mispredictions, techniques like reconstruction of mispredicted path [59] are used.

Trace-driven models do not include the run-time changes in behavior of multi-threaded applications [60]. This becomes a more visible problem if trace-driven simulation is run for a simulated multiprocessor system that is different from the one that was used to collect the trace. Trace-driven simulation should be avoided for parallel and timing-dependent systems as emphasized by Goldschmidt et al. [61].

Shade [62] is a trace-driven instruction set simulator, supporting SPARC and MIPS systems. Shade is also used to generate traces. Simplescalar also has the capability to run simulations from trace files. Cheetah [63] is a trace-driven simulator that simulates different cache configurations. MASE [64] is another example of this type of simulators. It is very hard for trace driven simulators to model the run-time changes in the behavior of multi-threaded applications [60], [61]. However, lately, few research works have been put forward to efficiently use trace-driven simulators for multi-threaded workloads, [65], [66].

2) EXECUTION-DRIVEN SIMULATORS

Execution-driven simulators do not use trace files. Instead, these simulators use binaries or executables of benchmarks for simulated target machines directly. These simulators can simulate misspeculated instructions unlike trace-driven simulators. However, they are complicated as compared to trace-driven simulators. SimpleScalar [27] falls into this category of simulators. Rsim [42], a discrete event-driven simulator based on YACSIM library [67], also interprets application executables rather than trace files. SESC and ESESC [68] are other examples of this type of simulators.

Often, users are interested in the performance of selected regions of code instead of entire benchmarks. The technique of *direct/native execution* can help in this respect. In direct execution, simulators only simulate particular portions of code (or *regions of interest*) of an application and execute the rest of the application directly on the host machine [57]. In this case, both the target and the host systems should have same instruction set architecture (ISA) to perform native execution. This technique is also referred as *co-simulation*. PTLsim [5] makes use of this method to speed up simulation. Tang [69], Proteus [70] and FAST [36] use this approach as well.

D. OTHER SIMULATOR CATEGORIES

Apart from the aforementioned classifications of simulators, simulators can also be classified based on other aspects or their specializations as follows:

1) MULTIPROCESSOR/MULTICORE SIMULATORS

Recently, multiprocessor/multicore systems have become ubiquitous. Multiprocessor simulators are more complex than uniprocessor ones as they have to cope with the challenges of

keeping the critical regions of applications consistent for all processors/cores and scheduling of processes [26]. Simulators with a modular design are better able to simulate multiprocessor systems, as they can easily instantiate different processors and corresponding modules to simulate a multicore system. There are two main approaches for simulating parallel targets: sequential simulation and parallel simulation. In case of sequential simulation, there is only one simulator thread to simulate all target cores. Simulators in this case simulate cores in a round robin fashion [26]. In case of parallel simulation methodology, different simulator threads are used to simulate different cores. This method speeds up simulation, but it is difficult to implement due to general challenges of multithreaded software development.

SimOS [54] and Simics [28] both support multiprocessor simulation. SimCA [71] is a simulator which is no longer maintained but was developed on top of SimpleScalar's out-of-order model. It focused on multithreaded processor architecture. MINT [41] is a software package that was designed to build event-driven memory hierarchy simulators for multiprocessors. It only runs on MIPS based machines and supports simulation of MIPS. PTLsim supports multithreading and multiprocessor simulation. Augmint [72] is a publicly available execution driven multiprocessor simulation environment for Intel x86 architectures. MINT forms the basis of Augmint, however; Augmint adapts Tango Lite's [73] augmentation technique as well. In this augmentation approach, the application is augmented with instrumentation code at compile time. This instrumentation code updates simulation clock and generates events for simulation. ZSim [7], ESESC [68], SESC [40] and SimCore [29] are other examples of simulators that support multicore simulations.

2) ENERGY AND POWER SIMULATORS

With the pressing need of building energy efficient processors and computer systems, the significance of energy and power simulators is increasing. There are many examples of such simulators in present days. Wattch [52] is widely used to simulate consumed power. It is based on Simplescalar and designed to examine and optimize power dissipation and energy consumption of Alpha microarchitecture. It can also be merged with other simulators. For example, SimWattch integrates Simics and Wattch. CACTI [74] is another example that simulates power and area for cache like structures. McPAT [17] can simulate timing, area and power of multicore processors. Some other examples are Powertimer [75], PowerAnalyzer [76] and SimplePower [77]. SESCTherm [78] and Hotspot [79] are two other examples of this category of simulators which model thermal effects at the micro-architectural level. Power Blurring [80] is another temperature calculating model, which is developed based on a matrix convolution approach, to reduce computation time. Ziabari et al. [81] have compared Power Blurring with HotSpot and SESCTherm. Their experiments have shown that the Power Blurring technique can achieve better accuracy to generate temperature profiles in less amount of time. ESESC [68] uses modified McPAT and Hotspot for energy simulations.

TABLE 1. Existing specialized/accelerator simulators.

Simulator type	References
Memory System	Mem-Sim [83], DRAMSim [84], Ramulator [25]
Cache only	Dinero IV [85], Cachesim5 [86]
Branch Predictor	[87]
Value predictor	CVPv6 [88]
Storage devices	HRaid [89], FlashSim [90], MQSim [91]
NoC	GARNET [92], NoC simulator [93], Noxim [94]
GPU	Multi2Sim [3], GPGPU-Sim [95]
ASIC	Aladdin [96], PARADE [97], Minerva [98], Firesim [99]
FPGA	ActiveHDL [100], ISim [101], Incisive Enterprise Sim. [102], ModelSim & Questa [103], QSim [104], FireSim [99]
DSP	[105], [106], [107]

3) SPECIALIZED/ACCELERATOR SIMULATORS

Many specialized simulators that are capable of simulating parts of a processor's architecture/microarchitecture exist. Among those, memory and network on chip (NoC) simulators are most common. Specialized simulators are only capable of executing certain types of instructions and usually use traces of specific instructions of executed programs/benchmarks as their inputs. They are easier to develop and can give a good idea about the behavior of specific parts of a processor; however, they are less accurate as they do not simulate the entire processor and the interaction with the other parts of the processor. They usually do not simulate 'off-path' instructions in case of mispredicted branches for example, in branch prediction simulators.

Memory simulators simulate data and instruction accesses to memory. Most of the existing memory simulators are trace-driven in nature, where trace files contain streams of memory accesses only. For example, DRAMSim is a timing simulator that can simulate different kinds of memories like DDR, SDRAM, DRDRAM etc. [83]. DRAMSim can also be integrated with other simulators. Cachesim5 [85] and Dinero IV [84] are examples of simulators that simulate only cache accesses. Network on chip simulators simulate the communication infrastructure of a processor. Emerging many-core processors design calls for a faster/less congested networks on a chip. As such, NoC simulator's have been increasingly built and used. They are capable of simulating different types of networks on chips, topologies, routing policies, etc.

Accelerator simulators have been used to simulate the behavior of programs or program portions accelerated using a graphical processing unit (GPU), an application specific integrated circuit (ASIC), a digital signal processor (DSP), a field programmable gate array (FPGA), near-data and in-memory processing, etc. Accelerators have been recently integrated with processors on the same chip or on a system-on-chip (SoC). In addition, they have been proposed to be tightly coupled with processors. The simulation of accelerators in addition to processors give a complete view of the performance of benchmarks. Table 1 shows different existing types of specialized and accelerator simulators with

TABLE 2. Comparison of different categories of simulation techniques in terms of accuracy, performance, level of details and easiness of development.

Simulation Model	Accuracy	Performance	Level of details	Easiness of development
Functional simulation	A-	P+++	L	E+++
Timing - cycle accurate simulation	A++	P	L++	E
Timing - event driven simulation	A+	P+	L+	E+
Coupled functional-timing	A+++	P	L++	E
Decoupled functional-timing/ timing first	A+	P	L+	E+
Decoupled functional-timing/ functional first	A	P++	L+	E++
Decoupled functional-timing/ timing direct	A++	P+	L++	E
Full-system simulation	A++	P	L+++	E
User-level simulation	A+	P+	L++	E
Trace-driven simulation	A+	P	L+	E+
executable-driven simulation	A++	P+	L++	E

Note: [evaluation parameter's first letter] with a suffix of +++ represents the highest value and without a suffix represents the lowest value

some examples. The interaction of accelerators and processors is also important to simulate, some computer architecture simulator's include (or can be integrated to) accelerator simulators such as gem5 [2] and Multi2Sim [3].

4) MODULAR SIMULATORS

Modular simulators, instead of having a monolithic design, contain independent modules for different portions of the processor that can be initialized and linked to other blocks of the simulated system. These simulators can be debugged easier and better suit complex designs than non-modular simulators. This modularity makes simulators more manageable. Liberty Simulation Environment (LSE) [107] is one example of modular simulators. LSE uses a single software function for each hardware component, and the designer can use those components connected in an hierarchy to construct any complex system. Asim [49] is another example of this type of simulators. It is a user-mode simulator that extends SimpleScalar to modular components within the simulator itself. MicroLib [21], M5 [46], Soonergy [108], [109] and gem5 are some other examples of modular simulators, which provide the ability to reuse modules of a certain processor component in a new computer system.

III. EVALUATION OF SIMULATORS

The evaluation of simulators is challenging because of the contradicting metrics that should be considered. Eeckhout [33] represented simulation trade-offs as a diamond with contradicting factors: accuracy, evaluation time, development time and coverage. In this paper, we consider six trade-off metrics that can be used to evaluate simulators: accuracy, performance, level of details, easiness of development, flexibility and user friendliness.

The accuracy of a simulator refers to the performance accuracy of the simulated target compared to real hardware. The performance of a simulator refers to how fast or slow the simulator can run while simulating the target architecture. The level of details represent the amount and level of details that a simulator includes while representing a target architecture. It is not easy to achieve the best results in all of these trade-offs as most of them are contradicting metrics. Flexibility refers to both the configurability of the simulator and how flexible the simulator is to modify (or add new) structures. User friendliness refers to how easy it is for users to learn how to use a simulator, modify it, and run different experiments. Table 2 compares the first four tradeoffs for the different categories of simulation models described earlier in section II. Flexibility and user friendliness are affected more by the simulation implementation than the category of simulators and thus excluded from the comparison.

IV. SUMMARY OF EXISTING PROCESSOR SIMULATORS

There exist several processor architecture simulators. Table 3 below summarizes existing computer architecture/ microarchitectural simulators; all simulators are open source except Simics. It summarizes different aspects of the simulators including supported hosts and targets, etc. In addition, it characterizes simulators based on the classification taxonomy described in section II above.

V. CHALLENGES OF SIMULATION

The main challenges in simulation are related to simulator's performance and accuracy [121]. This section describes these challenges and proposes strategies to tackle them.

A. SLOW SIMULATION

Computer architects and system designers rely on simulations with accurate timings for proper design decisions. Simulating a single application only can take a long time– from few hours to days. The primary reasons for long simulation time is the complexity of modern microarchitectures that are simulated and the length of today's programs, consisting of billions and trillions of instructions. With the advent of multiprocessor and multicore systems, simulators have to keep track of shared resources and deal with synchronization, which is resource consuming. Benchmarks have also become more complex than they were in the past [33]. For example, SPEC CPU benchmarks have become more complex overtime. The dynamic instruction count per benchmark was 2.5 billion on average in CPU89, it increased to 230 billion instructions in CPU2000 [122] and to 2.5 trillion instructions in CPU2006 [123]. CPU2017 has on average 10X higher dynamic instruction count compared to CPU2006 [124]. Today, applications are becoming increasingly multi-threaded to utilize multicore processors efficiently, and

TABLE 3. Simulators summary table.

Name	Supported hosts (ISA/OS)	Supported targets (ISA)	Category	Supported pipeline models	MultiCore Support	Notes
ASim [49]	x86	Alpha, x86	UM/MOD/TIM (decoupled timing and functional models)	out-of-order	yes	
Augmint [72]	x86/Unix, x86/Windows NT	x86	EDr/TD	—	yes	based on MINT and Tango Lite
CMP$im [35]	x86	x86	parallel UM cache (decoupled)	—	yes	uses Pin for functional simulation
COTSon [110]	x86	x86	FSys/FUNC	—	yes	
Dinero IV [83]	x86/Linux, Alpha/Linux, x86/Solaris, Alpha/OSF, SPARC/Solaris	input trace files	TD cache	—	no	integrated with many other simulators like Sim-Alpha, GEMS, MASE, MARSSx86
DRAMSim [84]	x86/Linux	input trace files	TD cycle-level DRAM	—	no	has power and thermal models, supports CPU-GPU simulation
ESESC [68]	x86-64/Linux and ARMv7	ARMv7	TIM/UM (cycle-level)	out-of-order, in-order	yes	uses Simics for full system simulation
Flexus [43]	x86/Linux	SPARC, x86	FSys/TIM/EDr (cycle-level)	in-order, out-of-order	yes	based on M5 and GEMS simulators
gem5 [2]	x86, ARM, SPARC, Alpha, PPC/ MacOSx, Solaris, OpenBSD	x86, ARM, MIPS, Alpha, PPC, SPARC	FSys/MOD/TIM (cycle-level)	in-order, out-of-order	yes	uses Simics for functional simulation, not maintained now
GEMS [38]	x86/Linux, AMD64-linux, and SPARCV9 (Solaris 8)	SPARC, x86	FSys/TIM (decoupled functional and timing models)	out-of-order	yes	supports Nvidia's Fermi and GT200 like GPUs, integrates energy model (GpuWattch)
GPGPUSim [16]	Linux	PTX and SASS, PTXPlus	UM cycle-level (decoupled timing and functional models)	in-odrer, out-of-order	yes	uses Pin for functional simulation, integrates power model
Graphite [47]	x86/Linux	x86	parallel UM/TIM (decoupled)	in-order with in or out-of-order memory completion	yes	based on Sim++
HASE [31]	x86/Linux, MAC, Windows	MIPS	MOD/FSys	out-of-order	no	
HAsim [111]	FPGA	MIPS	FPGA based TIM	out-of-order	yes	

Note: **FUNC**=functional, **TIM**=timing, **EvDr**=event-driven, **FSys**=full system, **UM**=user mode, **EDr**=execution-driven, **TD**=trace-driven, **MOD**=modular, **IS**=instruction set, μ**Ar**=micro-architecture, **HMP**=heterogeneous multiprocessor

TABLE 3. *(Continued.)* **Simulators summary table.**

Name	Supported hosts (ISA/OS)	Supported targets (ISA)	Category	Supported pipeline models	MultiCore Support	Notes
LSE [107]	x86/Unix	PowerPC, SPARC, IA64, DLX	MOD	out-of-order	yes	
LiveSim [112]	x86	MIPS64	TIM/UM (cycle-level)	out-of-order	no	based on QEMU and ESESC
MARSSx86 [4]	x86-64/Linux	x86-64	FSys/TIM (decoupled functional and timing models)	in-order, out-of-order	yes	based on QEMU and PTLSim
McPAT [17]	x86/Linux	Alpha, ARM, x86, SPARC	power, area and TIM	—	yes	flexible interface for easier integration with performance simulators
McSimA+ [60]	x86	x86	UM/TIM (decoupled functional and timing models)	in-order, out-of-order	yes (HMP)	uses Pin for functional simulation
MicroLib [21]	x86	Alpha, PowerPC, SHARC	MOD/FSys	out-of-order	no	modules are based on sysemC
Mint [41]	SGI, SPARC and DEC stations	MIPS	UM/EDr	—	yes	used by many other detailed simulators
MLRSim [56]	x86/Linux, SGI IRIX, SPARC/Solaris	SPARC v8	FSys/TIM (EvDr and MOD)	out-of-order	no	based on RSim
Multi2Sim [3]	x86/Linux	MIPS32, x86, ARM, AMD Evergreen, NVIDIA Fermi	UM/MOD/TIM	out-of-order	yes (HMP)	major use case is CPU-GPU simulation
MSim [37]	Linux/ x86, Win2000, SPARC/ Solaris	Alpha	UM/TIM (cycle-level)	in-order, out-of-order	yes	based on SimpleScalar, integrates Wattch's power model
OVPsim	x86/Windows, x86/Linux	ARM, MIPS, x86	FSys/FUNC	—	yes (HMP)	uses dynamic binary translation
PTLsim [5]	x86/Linux	x86	FSys/TIM (cycle-level)	out-of order	yes	processors like AMD K8, Intel P4 and Core 2 form the basis of core model, XEN integration for full system simulation
RSim [42]	SUN machines running Solaris 2.5, SGI Power Challenge running IRIX 6.2	SPARC v8	UM/EDr/EvDr/TIM	out-of-order	yes	based on MIPS R10000 architecture
SESC [40]	Unix-based systems (e.g. Linux and Darwin/MacOSx)	MIPS	UM/TIM/EvDr	out-of-order	yes	uses MINT emulator

Note: **FUNC**=functional, **TIM**=timing, **EvDr**=event-driven, **FSys**=full system, **UM**=user mode, **EDr**=execution-driven, **TD**=trace-driven, **MOD**=modular, **IS**=instruction set, μ**Ar**=micro-architecture, **HMP**=heterogeneous multiprocessor

TABLE 3. *(Continued.)* **Simulators summary table.**

Name	Supported hosts (ISA/OS)	Supported targets (ISA)	Category	Supported pipeline models	MultiCore Support	Notes
Shade [62]	SPARC	SPARC (v8 and v9)	profiler	—	no	supports dynamic profiling
SIMCA [71]	SPARC/ Solaris	Alpha, x86	UM/EDr/TIM	out-of-order	yes	based on SimpleScalar, not in active development now
SimCore [29]	x86/Linux, Alpha/Linux, UltraSPARC/Solaris, MIPS IRIX	Alpha	UM FUNC	—	yes	multiprocessing support at functional level only
SIMFLEX [44]	x86/Linux	x86, SPARC	FSys/MOD/TIM (decoupled functional and timing models)	out-of-order	yes	uses Simics for functional simulation and SMARTS for statistical sampling
SIMICS [28]	Alpha, PPC, UltraSPARC, x86/ Linux, Windows	Alpha, ARM, MIPS, PPC, SPARC, x86, Linux, Solaris, Windows	FSys/FUNC	—	yes	commercial simulator
SimOS [54]	x86/Linux, MIPS IRIX	SGI, IRIX MIPS	FSys/TIM	out-of-order	yes	based on MIPS R4000 & MIPS R10000, supports direct execution
SimpleScalar [27]	Linux/x86, Win2000/x86, SPARC/Solaris	Alpha, Pisa, ARM, x86	UM/EDr/TIM	out-of-order	no	models ranging from simple emulation to detailed simulation
SiNUCA [24]	x86-64/Linux	x86-64	TD/UM/TIM	out-of-order	yes	validated for Intel Conroe and Sandy-Bridge μ-Arch
Sniper [6]	x86/Linux	x86, **RISC-V**	parallel UM/TIM	in-order, out-of-order	yes (HMP)	uses Pin for functional simulation, based on interval simulation
SMTSIM [115]	Alpha/Unix, x86/Linux	Alpha	TIM	out-of-order	yes	
SPim [116]	Windows, Mac OSx, Linux	MIPS32	FUNC	—	no	used largely for teaching in academia
TEM2P2EST [117]	x86/Linux	Alpha, Pisa, ARM, x86	power/TIM (cycle-level)	out-of-order	no	based on SimpleScalar
Turandot [118]	AIX, Linux	PowerPC	UM/TIM	in-order, out-of-order	yes	
Wisconsin Wind Tunnel II [119]	SPARC	SPARC	parallel discrete EvDr/EDr	in-order	yes	part of Wisconsin Wind Tunnel Project
Zesto [120]	x86	x86	UM/TIM cycle-level	out-of-order	yes	built on top of SimpleScalar
ZSim [7]	x86/Linux	x86-64	parallel TIM/UM simulator	out-of-order, in-order	yes	

Note: **FUNC**=functional, **TIM**=timing, **EvDr**=event-driven, **FSys**=full system, **UM**=user mode, **EDr**=execution-driven, **TD**=trace-driven, **MOD**=modular, **IS**=instruction set, μ**Ar**=micro-architecture, **HMP**=heterogeneous multiprocessor

simulating multicore processorsf consumes more resources and time than single-core processors. Many techniques and innovative strategies have been proposed for accelerating the speed of simulation. Some of them are discussed below:

1) SAMPLED SIMULATION

One of the mostly used techniques to accelerate simulation is Sampling. In sampled simulation, instead of simulating the entire benchmark only a small number of samples are simulated. These samples are groups of instructions, which are considered to represent the entire benchmark. The selection of the sampling points can be done in two ways: (1) statistical sampling and (2) targeted sampling. One approach for statistical sampling is to randomly pick samples from the entire instruction stream to acquire unbiased samples. The other statistical sampling approach is to go for periodic sampling which selects sampling units at regular intervals across the entire program. For example, periodic sampling is used by SMARTS [20] and Flexus [43].

Targeted sampling picks sampling points after analyzing program's behavior. Single sampling points/units are selected from each phase (a phase is a group of a large number of consecutive instructions that have a similar behavior). The weights of phases are usually calculated and considered in choosing sampling points. Since target sampling uses program behavior to pick sampling points, it is possible that the targeted sampling may result in less number of total samples compared to statistical sampling. However, target sampling cannot provide a confidence bound on performance estimates [33]. SimPoint [125] is a tool that follows the targeted sampling approach and uses basic blocks to detect program phases. SimPoint combines basic blocks into intervals, then uses Manhattan distance to find the similarities among different intervals to locate program phases; each phase contains many intervals [126]. Intervals from each phase are chosen as a sampling point for simulation to represent a complete picture of program execution. Each sampling point is referred as a simulation point or sometimes a *SimPoint*. Yi *et al.* [127] performed a comparison of Simpoint and SMARTS. Their study indicated that SMARTS is more accurate but slower than SimPoint.

There are two challenges associated with sampled simulation techniques [33], [128]. The first challenge is to accurately provide a sampling point with its architectural state's starting image (ASSI). Functional simulators require ASSI, the processor's architectural state (register and memory contents), for each sampling point to achieve a correct output. In addition, timing (performance) simulators use ASSI checkpoints for accurate timing simulation, i.e., the ASSI should be as close as possible to the program's architectural state at the beginning of the simulation point to achieve accurate simulation results. Many simulators have a *fast forwarding* feature that uses a quick functional simulator to construct the ASSI [33], [128]. Timing simulators use detailed simulation for the entire sampling point, but switches to functional simulation at the end of each sampling point until it reaches the next one. Fast forwarding can consume a considerable amount of time when the sampling units are located far from the start of the program and far from each other in the dynamic instruction execution [33]. In addition, using fast forwarding and detailed simulation intervals serializes the simulation for all the sampling points. This means, to construct the ASSI for the next sampling unit, one needs to simulate all previous sampling units and fast forward between the sampling units. One technique that can be used to speed up fast forwarding is *direct execution* as implemented in PTLsim [5], where the program is executed on native hardware directly instead of functional simulation. *Checkpointing* [129] is another technique that can be used to solve this problem of initiating an architectural state [33]. Checkpointing stores the ASSI up until a sampling point. This checkpoint is then loaded from a disk during sampled simulation. Checkpointing also allows parallel simulation as opposed to fast forwarding [33]. However, the space required to store large checkpoint files on disk is a drawback associated with checkpointing.

To achieve high accuracy, sampled simulation also requires an accurate starting image for microarchitectural state that contains the state of branch predictors, caches, etc. Various strategies for cache state warmup are used [33], such as: continuous warmup, cache miss rate estimation, self monitored adaptive warmup, boundary line reuse latency, and checkpointing. A processor's core structures such as functional units, reservation station, reorder buffer, etc. also need to be warmed up to achieve highest accuracy [33]. For large sampling units, this is not a crucial problem as a processor's core does not keep long history of events as branch predictors and caches do. However, for short sampling points, it can be important to accurately warmup these structures.

One of the problems associated with the sampling techniques is their inability to work with multi-threaded applications. Tools like Simpoint [125] and SMARTS [20] do not support multi-threaded workloads. However, recently some research have attempted to resolve this problem for specific classes of multithreaded applications. For example, BarrierPoint [130] leverages the synchronization barriers in multithreaded applications to sample a number of iterations in a workload between two barriers.

2) STATISTICAL SIMULATION

Statistical simulation [131] combines detailed and analytical simulation. First of all, a statistical profile containing important program characteristics is computed using simple trace-based tools. The statistical profile is used to create an instruction trace that can be fed to a trace driven simulator. These synthetic traces are very small in size and simulation process can proceed quickly. Because statistical simulation may not be accurate enough for architects to make design decisions, it cannot replace detailed simulation [132]. However, it is useful to recognize interesting regions in programs for additional analysis. Using statistical simulation, an average error in IPC values can range from 10% (for simple cores) to 15% (for aggressive cores) [131].

However, for design space exploration, a performance model's relative accuracy (relative accuracy refers to the accuracy in observed performance changes obtained by changing microarchitectural parameters) can be more important than its absolute accuracy because often we are only interested in knowing the performance impact of a change in a particular parameter. Eeckhout et al. [131] have tested statistical simulation using SPECint95 benchmarks. Their results show that the relative IPC error is below 0.9% for various cases. Statistical simulation has also been applied to power modeling and system evaluation, for example as in Wattch.

3) PARALLEL SIMULATION

Parallelizing simulators can significantly reduce the simulation time for each run. If checkpointing is employed in sampled simulation, parallelism can be applied to simulate multiple sampling points at the same time; this is called *parallel sampled simulation* [33]. Secondly, to make use of ubiquitous multicore processors, it seems tempting to come up with simulators that are multithreaded and can make use of parallel processors. The simulator's code can be divided into different threads, where each thread can be mapped to each target core in case of simulating multicore architectures. One issue with parallel simulation is the balancing of speed vs accuracy [33]. For cycle-level simulators, threads would need to synchronize after every cycle, which can potentially be a barrier in performance gain expected by using parallelism. As a solution, the condition of cycle-by-cycle synchronization can be relaxed to more than one cycle as a tradeoff between accuracy and simulation speed [33], [133]. BigSim [134] uses parallel simulation, to simulate machines with a large number of processors. Sniper [6], Graphite [47], Barra [32] and ZSim [7] are other examples of parallel simulators.

4) FPGA ACCELERATED SIMULATION

To speed up the process of simulation, parts of simulators can be implemented on a Field Programmable Gate Array (FPGA). Simulators can leverage the fine grained parallelism available on FPGAs to achieve higher simulation speeds compared to pure software simulators [36]. However, the development time of an FPGA-based simulator can be large compared to software simulators, as FPGA accelerated simulators have to be written in hardware description languages (HDL). They are also not as parametrizable as software simulators. HAsim [111] is a timing-directed execution-driven simulator, which implements both functional and timing models on an FPGA. Another example is FAST [36]; it is based on the functional-first simulation strategy. FAST's functional simulator is implemented in software, while its timing simulator is a hardware based simulators and run on FPGAs. Recently, there has been a RISC-V Chisel-to-Verilog simulator converter that converts a simulator written in a new hardware construction language developed by UC Berkeley, Chisel, to Verilog HDL [135]. Then processor simulation can be run directly on FPGAs. Writing Chisel code can be less time consuming than writing a Verilog code; however, it requires learning a new language. Similarly, Fabscalar is an x86 simulator, written in a HDL/C++, which allows users to work with synthesizable parameterized register-transfer-level (RTL) to simulate x86 designs using FPGAs [136]. However, the simulator is not very configurable as it only allows the user to choose among different structures and supported parameters that are already implemented in HDL.

B. POOR ACCURACY

Fidelity of simulation should be a serious concern, considering the reliance of major design decisions on simulation results. First, simulator developers have to make sure that simulators are functionally correct, if they are simulating a target's functionality. Second, the performance statistics should indicate the target's actual performance. Unfortunately, simulators are not always accurate and can exhibit various errors. Potentially, there can be three different types of errors in simulators [137]: Modeling errors, specification errors and abstraction errors.

Modeling errors occur when the desired functionality is not properly implemented or modeled in the simulator. One example of modeling errors is when instructions are configured to take different latencies than the modeled target. Another example can be issuing instructions to reservation stations in an out-of-order manner. Modeling errors can be reduced by carefully designing and testing the modeled structures. Errors can be further reduced using proper design strategies and software engineering principles.

Second, specification errors result from to the lack of knowledge about the correct functionality of the target. Specification errors can only be decreased if the target's specifications documentation is accessible. If certain specifications of the real hardware are not known, writing microbenchmarks can help estimating some specifications. For example, one can estimate the size of the reservation station by writing and running a microbenchmark for different cases.

Third, abstraction errors occur when developers implement their design at a higher level of abstraction to tradeoff design details for a better speed, or to simplify their simulator's implementation. To reduce abstraction errors, developers usually tradeoff speed; simulator writers can reduce abstraction errors by including more details in their simulation models. Today's new technologies with faster hardware enable further reduction of abstraction errors. Cain et al. [138] discuss some sources of abstraction errors that affect simulation accuracy. First, they conclude that OS effects are important, thus, going for full-system simulation can make simulation more accurate and representative of true behavior of the target, and reduce abstraction errors. They also found that the simulator's accuracy can be affected by simulating the I/O behavior even for uniprocessors. Another example of an abstraction error is not simulating incorrect speculative paths, which can reduce the accuracy of the simulator. However, for certain

commercial applications and SEPC CPU integer benchmarks, it was shown that simulating these incorrect paths affected performance by only 2% [138].

VI. VALIDATION OF SIMULATORS

Simulator validation refers to the process of validating that a simulator accurately represents a target hardware. Simulator validation is important to ensure that a simulator does not include modeling, specification or abstraction errors. Validating simulators usually incorporates modeling a real hardware and comparing the simulators results to those of the real hardware, then calculating the experimental error. If the experimental error is high, then first the types and sources of errors should be identified. After that the simulator should be modified to correct those errors. This process can be repeated until an acceptable experimental error is reached. Validated simulators give confidence to users in their results and is important for result reproducibility. Gibson *et al.* [139] and Black and Shen [137] recommended that simulation studies should be compared against a reference hardware platform or an already validated simulator. A recent study [140], which calibrates MARSSx86 simulator for a particular target heterogeneous processor, concludes that an unvalidated/uncalibrated simulator can lead to considerable differences between simulation results and real architecture performance statistics.

While validation of simulators is important before relying on their results, some researchers view that rigorous validations are not always possible and unvalidated simulators can still give deep insights into design decisions [42]. Hughes *et al.* [42] argue that validating simulators can be impractical for research and many unvalidated simulators can prove useful and valuable in studying architectural phenomenon and relative performance. It can be a tedious job to validate a simulator due to: (1) the lack of certain details about modern processors where it becomes almost impossible to implement those systems precisely in simulators; (2) implementing some details of a modern processor, even when known, can be time consuming and can result in a slower simulation time; (3) modeling a target system that is just a a research idea, thus it is hard to validate. In such cases, it becomes impossible to validate simulators [33].

We can find many validation efforts for various simulators in the literature [14], [19]–[24], [60], [68], [140]–[144]. These validation efforts differ in the strategies they used. Mostly, computer architects compare the results of their simulators with the performance behavior of benchmarks on the real machine that is being simulated. Sometimes, they also rely on published results for a particular hardware instead of running experiments on the real system. Desikan *et al.* [19] tried to validate SimpleScalar's out-of-order model against Alpha 21264 processor model. The mean experimental error in microbenchmark simulation was reduced from 19.5% for sim-outorder, to 2% after validating sim-outorder for an Alpha processor (*Sim-alpha*). *Sim-alpha* [145] is based on SimpleScalar simulator and uses code from sim-outorder, but almost all of the timing simulation model is written from scratch. For macro-benchmark validation (benchmarks are taken from SPEC-CPU2000 [146] suite), the sim-outorder resulted in an average experimental error of 36.7% compared to DS-10L (alpha processor based machine) [19]. ESESC [68], which is an extension of SESC [40] simulator, is validated against Samsung Chromebook (contains ARM A15) using SPEC CPU benchmarks. They used *perf* utility [147] on Chromebook with performance counters enabled, to collect statistics on the real system. Their results show 21% IPC (instruction per cycle) error on average. Perez et al. [21] compared performance estimation of various modules of MicroLib simulation environment with SimpleScalar to validate them. Walker *et al.* [144] proposed a new method to find sources of inaccuracies in simulators and validate them using clustering, correlation analysis and regression. Our previous work [148] compared the experimental error of few computer architecture simulators to an Intel's Core-i7 microarchitecture. A recent work by Jo *et al.* [149] introduced*DiagSim* to detect the hidden details in three simulators (gem5, Multi2Sim, MARSSx86) that can impact simulation results significantly. In addition to the simulators survey part, this paper discusses more examples of validation efforts for recent simulators in Section VII and *further compares* their absolute and relative performance in Section IX.

VII. COMPARISON OF RECENT X86 SIMULATORS

We selected six simulators: gem5 [2], MARSSx86 [4], Multi2Sim [3], Sniper [6], PTLsim [5] and ZSim [7] for a comprehensive study due to following reasons:

- These simulators have different simulation models, but all of them fall into the category of timing simulators.
- These are modern simulators with active development, except PTLsim. PTLsim is not in active development, but it is still used today.
- All these simulators support x86 and other major architectures. They also have the ability to perform detailed simulation on selected parts of any benchmark.

A. gem5

gem5 [2] is an event-driven full-system simulation tool, which is extensively used in both academia and industry. Although gem5 is an event driven simulator, it can keep track of events on a cycle-by-cycle basis, which makes its accuracy comparable to a cycle-level simulator. It supports many ISAs: ARM, x86, MIPS, SPARC, ALPHA, Power and RISCV. It uses CPU models from M5 [46] and memory system models from GEMS [38]. There are mainly four CPU models in gem5: 'AtomicSimple', 'TimingSimple', 'Minor' (in-order) and 'O3' (out-of-order). The first two models (AtomicSimple and TimingSimple) are single-cycle processor models without any pipelined structures. AtomicSimple models the timing of memory accesses but TimingSimple does not. Minor and O3 are 'execute-in-execute' pipelined models. These models allow for configuring

multiple pipeline stages and their widths, functional units, and other pipeline structures. The 'O3' model supports simultaneous multithreading (SMT). Recently, kernel-based virtual machine CPU (KVM-CPU) model was also introduced in gem5 that allows the simulated code in full-system to run on the actual hardware, thus increasing the simulation speed significantly. This CPU can be used for fast-forwarding through the non-important parts of the simulated code.

Gutierrez *et al.* [22] and Butko *et al.* [141] evaluated gem5's accuracy to model actual processors based on ARM ISA (Cortex A15 and A9 microarchitectures). Gutierrez *et al.*'s experiments showed an average inaccuracy of 13% for SPEC CPU2006 benchmarks [22]. Butko *et al.* [141] studied gem5's accuracy for multicore embedded target's simulation. Tanimoto *et al.* [150] also pointed some of the issues with the out-of-order implementation of gem5. Walker et al. validated gem5 against two ARM microarchitectures [144]. The inaccuracy varied from 1.39% to 17.94% based on their experiments. Akram and Sawalha calculated the experimental error for gem5 with x86 ISA and pointed out some sources of inaccuracy [148]. However, there is no full validation effort for x86 ISA.

B. MARSSx86

MARSSx86 is an x86 full-system simulator [4] that is modeled at the cycle-level. The detailed pipeline model of MARSSx86 is based on PTLsim [5]. In addition, various optimizations for better performance and flexibility were added. MARSSx86 uses QEMU [151] based full-system emulation environment to perform full-system simulation of unmodified operating systems. It supports both out-of-order and in-order (IO) pipeline models. MARSSx86 allows for the simulation of heterogeneous configurations. It also supports real time input/output devices' simulation.

Asri *et al.* [140] calibrated MARSSx86 to simulate an Intel Core i7 machine, with a focus on high performance computing applications. Their study exposed certain issues (e.g. overestimated number of μ-ops, when decoding instructions to μ-ops) with the simulator. The final calibrated MARSSx86 simulator is shown to have less than 10% error on average for SPEC and PARSEC benchmarks.

C. Multi2Sim

Multi2Sim is a simulator that mainly targets GPUs and simulates CPU-GPU architectures [3]. It supports many ISAs for example, x86, MIPS, ARM and AMD Evergreen ISA. Multi2Sim mainly consists of three different simulation blocks: a functional simulation engine, a detailed simulator, and an event-driven module. The detailed simulator and the event-driven module together perform timing simulation. It supports multi-threaded or single-threaded processor cores with an out-of-order (OoO) pipeline. It does not model IO pipelines. Memory and interconnection networks can be configured with good flexibility. Multi2Sim follows the design philosophy of Simplescalar [27] for some of its modules. Moreover, it is a timing first simulator like GEMS [38]. Multi2Sim does not support simulation of an entire operating system, but it can use dynamic threads to simulate parallel programs.

Multi2Sim's validation for GPUs has been done by Ubal *et al.* [142]. They used AMD Radeon 5870 as a target GPU model and AMD OpenCL SDK [152] applications for benchmarking. The results verified the functional correctness in addition to measuring the average percentage error in execution time (5% to 30%). To the best of our knowledge, there are no validation efforts for x86 CPUs for this simulator.

D. PTLsim

PTLsim [5] is a cycle-level simulator that has the ability to simulate complete OS using Xen hypervisor [153]. It makes use of co-simulation or a direct execution technique, which has been discussed previously. It is capable of modeling a superscalar OoO core. It does not model a detailed IO pipeline. PTLsim's default core model (OoO superscalar) is based on characteristics of different real systems like Intel's P4 and Core 2 processors and AMD's K8 processor.

Yourst [5] has evaluated the accuracy of PTLsim. He used a real machine with 2.2 GHz AMD Athlon 64 processor as a reference. He used rsync [154], which is a client server application, as a test benchmark. The results show that PTLsim's inaccuracy in many cases is less than 5%.

E. SNIPER

Sniper [6] is a fast parallel simulator that uses the interval simulation method discussed earlier [45]. Sniper is based on Graphite [47], which supports various one-IPC models. Sniper supports both OoO and IO pipeline simulation. Carlson et al. [6] validated Sniper using Intel Xeon X7460 machine. They showed an inaccuracy average less than 25% for SPLASH-2 benchmark suite. Later, to further improve the accuracy of sniper, Carlson *et al.* [23] implemented an instruction-window based model in Sniper. The improved simulator exhibited a single-core inaccuracy of 11.1% compared to an Intel's Nehalem based target system. Originally, Sniper supported x86 only, however, recently a support for RISC-V ISA has been added to the simulator [155].

F. ZSim

ZSim [7] is a parallel application-level timing simulator for x86-64 architectures. It was initially written to model ZCache [156], but has grown into a more resourceful simulator. It focuses more on simulating memory hierarchies and many core heterogeneous (single-ISA) systems. It supports modeling both OoO and IO pipelines. Extensive use of dynamic binary translation allows it to achieve very high simulation speeds. ZSim's validation [143] using an Intel Westmere core showed an error of 10% on average. Average absolute error for multi-threaded workloads is 11.2%. Different microbenchmarks, single threaded (SPECCPU2006) and multi threaded (PARSEC, SPLASH2) benchmarks were used for the validation effort. The validation study shows that

TABLE 4. Feature comparison (updated version from [148]).

Feature	gem5	Sniper	PTLsim	Multi2Sim	MARSSx86	ZSim
Host support	H++	H	H	H+	H	H
Target support	T++	T	T	T+	T	T
OS simulation	yes	no	yes	no	yes	no
Fast forwarding	yes	yes	no	yes	yes	yes
Trace creation	yes	yes	yes	yes	yes	yes
Checkpointing	yes	no	no	yes	yes	no
Details of results	D++	D	D+	D+	D+	D
Pipeline length option	yes	no	yes	no	yes	yes
Power/energy simulation	P+	P	P	P	P	P
IO core	yes	yes	no	no	yes	yes
HMP option	M,G	S	no	M,G	S	S
Support for GPUs	yes	no	no	yes	no	no
Support for parallel apps	yes	yes	yes	yes	yes	yes
Forum support	F++	F++	F-	F	F+	F+
Note: M=Multi-ISA, G=GPU, S=Single-ISA, IO=In-order, [parameter's first letter] with a suffix of ++ means it is better than a suffix of +, which is better than without any suffix, which is better than a suffix of -						

the main sources of errors in the simulator are the idealized branch target buffer and the inability to model translation lookaside buffers (inaccuracies in the front end model of the pipeline).

G. FEATURE COMPARISON OF SELECTED SIMULATORS

In our previous work [148], we compared the features of some x86 simulators discussed above. In this paper, we add one more simulator to our detailed comparison, MARSSx86, as shown in Table 4. gem5 can run on the highest number of OS's (e.g. Linux, MacOS X, Solaris, OpenBSD) and architectures (e.g. x86, x86-64, ARM, SPARC, Alpha and PPC) in comparison to the other simulators. Multi2Sim supports Linux, MacOS X machine with x86. Sniper, PTLsim, MARSSx86 and ZSim run on a Linux based x86 machines.

All these simulators support fast-forwarding and cache warmup except PTLsim. Sniper is not capable of creating checkpoints by itself but makes use of Pin [34] and Simpoint tools for checkpoints creation. All these simulators can create execution traces during simulation. Complex out-of-order pipeline simulation in detailed mode is supported by all simulators, but IO pipeline is not supported by all of them such as Multi2Sim and PTLsim. gem5 produces very detailed simulated performance statistics (e.g. block and idle cycles of all pipeline stages, squashed instructions at different stages due to branch mispredictions and memory order violations). Multi2Sim, PTLsim and MARSSx86 also produce detailed statistics but the details are less than those produced by gem5.

In gem5, MARSSx86, PTLsim and ZSim, the penalty of branch mispredictions can be changed by changing the pipeline depth. On the other hand, changing the penalty of branch mispredictions in Sniper and Multi2Sim can be done by directly specifying misprediction penalty and instruction latencies. Sniper and gem5 support dynamic voltage and frequency scaling (DVFS) to study runtime effects on energy efficiency. Statistics from all these simulators can be used to derive power/energy models like McPAT [17]. In terms of heterogeneous multicore (HMP) simulation support, ZSim and Sniper support simulation of HMP systems with only a single ISA, where the HMP processor can have different core parameters like execution models, frequency, dispatch widths, window sizes, etc. gem5 has been currently integrated with GPUsim to model CPU-GPU heterogeneous simulations [157]. Moreover, gem5's code can be slightly changed to support the simulation of multi-ISA HMP. Multi2Sim integrates CPU and different GPU architectures that can be used for CPU-GPU simulation. In terms of community and support forums, Sniper and gem5 have decent sized support groups. Multi2Sim, ZSim and MARSSx86 also have such forums; however, the support forum for PTLsim is no longer continued.

VIII. METHODOLOGY AND EXPERIMENTS

This section discusses in details the experimentation methodology adopted to compare the results of the six selected simulators with that of real hardware results, and find the experimental errors.

A. THE TARGET SYSTEM

The target system used for our experiments is based on an Intel's Haswell microarchitecture (core i7-4770). While all configuration parameters are not published by Intel for this system, we had to rely on other sources to configure the simulators to match this target. We used both Intel documentation [158] and some other resources [159]–[161] to configure the simulators to model Haswell. We used the same features of the target as our prior work [148], see Table 5. It should be noted that these simulators do not support micro-operation (μ-op) fusion, so the width of pipeline stages is set to a comparable number of simple μ-ops (for example an

TABLE 5. Target configuration.

Feature	Core i7 Like
Pipeline type	OoO
Number of stages	19
Width of fetch stage	6 instructions
Width of decode stage	4-7 fused μ-ops
Size of decode queue	56 entries
Width of issue/rename stages	4 fused μ-ops
Width of dispatch stage	8 μ-ops
Width of commit stage	4 fused μ-ops
Size of reservation station	60 μ-ops
Size of reorder buffer	192 μ-ops
L1 dcache associativity, size	8 way, 32KB
L1 I$ associativity, size	8 way, 32KB
L2 cache associativity, size	8 way, 256KB
L3 cache associativity, size	16 way, 8 MB
Latency of L1 cache	4 cycles
Latency of L2 cache	12 cycles
Latency of L3 cache	36 cycles
Size of cache block	64 Bytes
Instruction latencies	[160], [161]
BTB size, associativity	4096, 4 way
RAS size	16 entries
Branch misprediction penalty	14 cycles
Integer/Floating physical registers	168/168
Instruction TLB size	128 entries
Data TLB size, associativity	64 entries, 4 way
L2 TLB size, associativity	1024 entries, 8 way

issue width that is four fused operations, is set as six simple operations for our experiments).

We configured the simulators to match the chosen reference system to the best of our knowledge. We also made sure that the configurations across the simulators are similar to each other, with only minimal changes to the code of simulators, so that a fair comparison can be made based on simulation results. The purpose of this study is to compare the simulators' absolute and relative accuracy to each other, not to validate simulators. Thus, we did not try to implement new hardware structures or optimizations that certain simulators do not support. Table 6 describes few configuration parameters, which are not same across all simulators or are different from Haswell configuration due to limitations of simulators' support [162]. For example, the specifications of Haswell branch predictor are unknown and the supported branch predictors in the studied simulators are not exactly the same. As shown in Table 6, we configured branch predictors as close to each other as possible. gem5's tournament branch predictor is based on Alpha 21264 machine [163] and uses a local and a global history tables along with a choice predictor. Sniper uses a branch predictor that is modeled after Intel Pentium M's branch prediction unit. This predictor is identical to McFarling's serial BLG predictor [164] and uses loop, bimodal and global predictors as well [165]. To support this we changed some hardcoded configurations in Sniper. Sniper also contains a 256-entries indirect branch target buffer (iBTB) [165]. PTLsim has many options related to branch predictor configuration. A hybrid bimodal and G-share predictor is configured for these experiments. The same branch predictor is used for MARSSx86 as well. A tournament branch predictor containing a bimodal and a two level predictor, is configured for Multi2sim. The sizes of individual predictors are shown in Table 6. PTLsim supports both partitioned and shared instruction issue queues [148].

PTLsim does not allow for modeling of shared instruction queues and clusters at the same time. Therefore, following the example of [166], we configured partitioned instruction queues in PTLsim with extra entries to account for any performance loss caused by partitioned queues. The details of cache prefetching structures are not known for Haswell, so we deactivated prefetching on real hardware and also did not configure them on the simulators. The details of all of the used configurations for all simulators can be found in our technical report [162].

B. EXPERIMENTAL WORKLOADS AND PERFORMANCE MEASUREMENT ON REAL HARDWARE

We used SPEC-CPU2006 [167] and a subset of MiBench [168] embedded benchmarks. The embedded benchmarks can complete their execution in a realistic time on the simulated system, but a complete execution of SPEC benchmarks can take a very long time. Thus, we ran each SPEC benchmark for 500 million x86 instructions. These instructions were chosen from a representative segment of the program using Simpoint [125]. Also, a warmup period of 100 million instructions was used.

On the real hardware, we used PAPI [169] to measure instructions per cycle (IPC), cache misses, branch mispredictions values for the entire execution of embedded benchmarks. For SPEC-CPU2006 benchmarks we measured the same parameters for the same 500 million simulated instructions. In order to eliminate the effect of system perturbations on the real hardware event measurements, we ran the benchmarks multiple times in a non-continuous manner and then calculated the average of all runs. The standard deviation in IPC results for all runs is on average 0.02265. We used gcc 4.4.7 compiler to compile the different benchmarks. We generated both 32-bit and 64-bit binaries of the benchmarks for our experiments (depending on what each simulator supports). Specifically, we used 32-bit binaries for Multi2Sim, PTLsim and Sniper (also used 64 bit binaries with Sniper), while we used 64-bit binaries for the rest of the simulators. The same binaries were used for the simulators and the real hardware runs. The host operating systems used for building and running of workloads are Scientific Linux 2.6.32 (32 bit) and Ubuntu 14.04 (64 bit) for 32-bit and 64-bit binaries respectively. The reference hardware's performance counter values were calculated on both hosts. This work uses gem5's

TABLE 6. Differences in simulator configurations.

Parameter	Gem5	Sniper	PTLsim	Multi2Sim	ZSim	MARSSx86
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)	Hybrid (Bimodal:4K, Gshare: 4K Choice: 4K)
BTB associativity	1 way (direct)	4 way	4 way	4 way	ideal btb	4 way
TLB associativity	1 way (direct)	4, 8 way	1 way	-	no tlb	1 way
Instruction queue	unified 60 entries	unified 60 entries	distributed 4x16 entries	unified 60 entries	unified 60 entries	unified 60 entries
Memory address disambiguation	store sets	NC	NC	NC	NC	NC
Delay between pipeline stages	configured to get 19 stages	NC	NC	NC	configured	NC
TLB levels	one	two	one	NC	no TLB	one
Note: NC = Not Configurable						

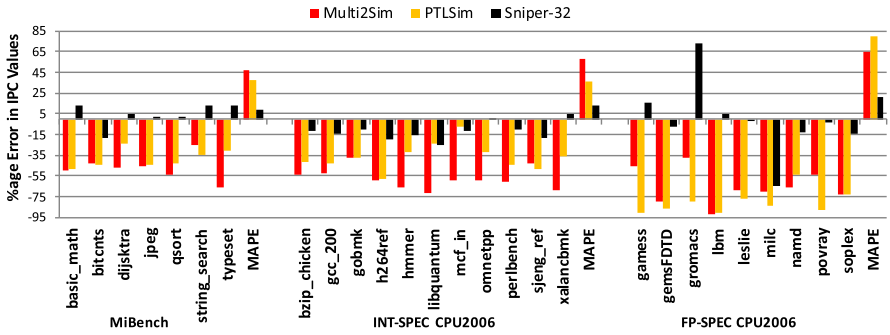

FIGURE 5. Percentage error in IPC values for 32-bit binaries.

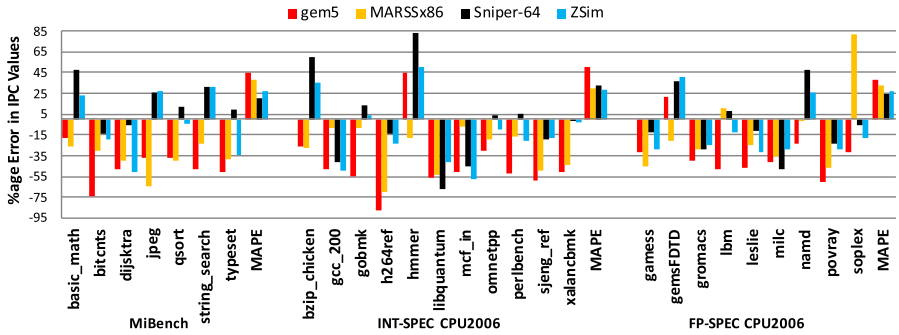

FIGURE 6. Percentage error in IPC values for 64-bit binaries.

stable version of September 2015, MARSSx86 version 0.4, Multi2Sim version 5.0, Sniper version 6.0, ZSim's stable version of April 2016 and PTLsim version available at [170] for all experiments.

IX. RESULTS

We simulated previously mentioned benchmarks on the six different simulators and compared their simulation results with the real target hardware results. We calculated the experimental error of the simulators against real hardware runs. We also performed a sensitivity tests to find the effect of changing certain configurational parameters of each simulator compared to other simulators.

A. ERROR ANALYSIS

Figure 5 and Figure 6 show the percentage error in IPC values for all benchmarks on all simulators when compared to IPC values of benchmarks from the reference hardware runs. The mean absolute percentage error (MAPE) in IPC values is the lowest for Sniper for all categories of benchmarks.

To study the sources of errors that cause the observed inaccuracy in IPC values, we looked into cache misses and

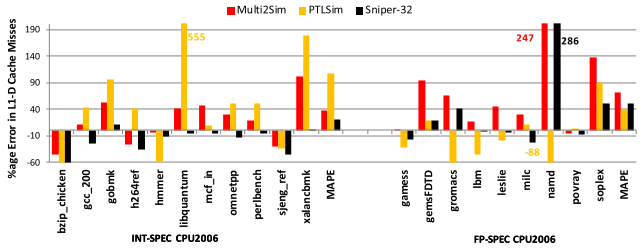

FIGURE 7. Percentage error in L1 DCache misses for 32-bit binaries.

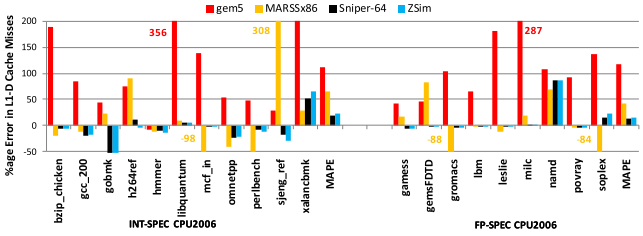

FIGURE 8. Percentage error in L1 DCache misses for 64-bit binaries.

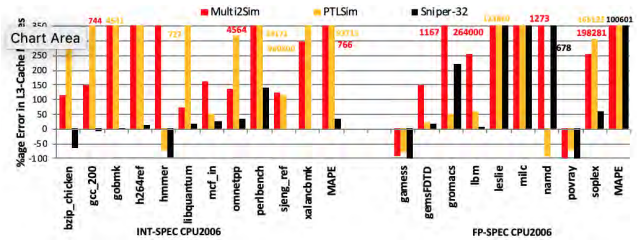

FIGURE 9. Percentage error in L3 cache misses for 32-bit binaries.

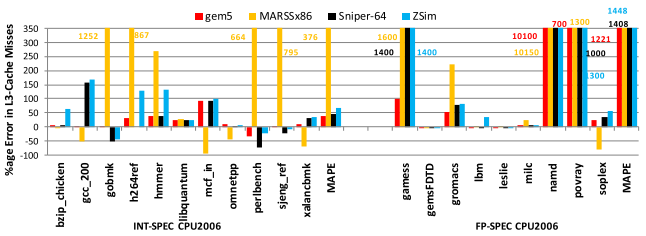

FIGURE 10. Percentage error in L3 cache misses for 64-bit binaries.

branch mispredictions for SPEC benchmarks on these simulators. Figures 7 and 8 show the percentage error in L1 data cache misses, Figures 9 and 10 show the percentage error in L3 cache misses and Figures 11 and 12 show the percentage error in the number of mispredicted conditional branches shown by the different simulators. The percentage error in these statistics is very high for some benchmarks (much higher than 100%) as shown in the figures. On average, FP-SPEC benchmarks show higher error rate in cache misses compared to INT-SPEC benchmarks as they consist of larger numbers of memory instructions, and INT-SPEC benchmarks show higher error rate in branch prediction accuracy as they consist of larger numbers of branch instructions.

PTLsim showed a high inaccuracy in the floating point benchmarks; several benchmarks showed an inaccuracy above 50%. The main reason we found for this high underestimation of IPC in PTLsim is related to decoding x86 instructions into μ-ops. The benchmarks which show highly inaccurate IPC values, exhibit high ratio of μ-ops to x86 instructions for PTLsim compared to the other simulators. For instance, this ratio for *gemsFDTD*, *gamess*, *povray* and *soplex* is 9, 6.07, 5.43 and 4.2 respectively on PTLSim. On gem5 and Multi2Sim, the observed μ-ops to x86 instructions ratio for the same benchmarks is always less than 2.30. Since, pipeline width in these simulators is defined by number of μ-ops, high values of μ-ops to x86 instructions ratios affect the performance of the pipeline.

There are many examples in Figures 7 and 12, which indicate more than 100% inaccuracy in L1 data cache misses and branch misprediction respectively. These figures assist in understanding negative errors in IPC numbers for few applications. For instance, many of the benchmarks run on gem5 (*gobmk*, *gcc_200*, *h264ref* , *perlbench*, *povray*, *namd*) exhibit overestimated branch misprediction rate and data cache misses in comparison to the reference target architecture. For these benchmarks, the simulator's branch predictor

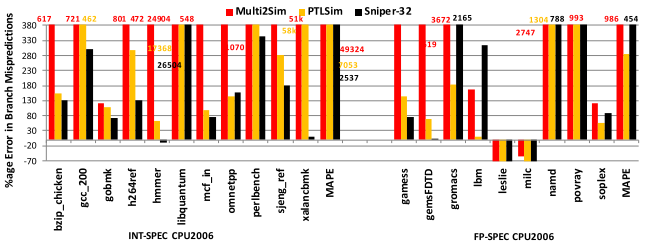

FIGURE 11. Percentage error in branch mispredictions for 32-bit binaries.

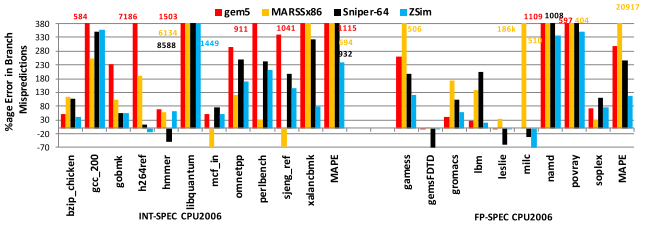

FIGURE 12. Percentage error in branch mispredictions for 64-bit binaries.

does not emulate the behavior of the actual core's branch predictor, which results in a higher inaccuracy. When the benchmarks with a very high number of branch predictor misses are compared to the benchmarks with a lower number of branch predictor misses, it is observed that they contain a much higher number of branch instructions (20% or more of the overall instruction mix). This high count reveals the inadequacy of simulator's branch predictor to model the target's (Haswell) branch predictor for those benchmarks.

gem5's way of decoding and implementing some of the x86 instructions can explain some IPC inaccuracies. An example is an integer divide operation, which is decoded into many μ-ops, where each μ-op is responsible for calculating one quotient bit. The hardware division algorithm on the real system is different from the simulator's implementation. Moreover, the mislabeled μ-ops in gem5 cause some

inaccuracies in the results [15]. Another problem observed with gem5 is related to the throughput of its fetch stage. The current implementation of the fetch stage does not allow initiating new requests when it is waiting for a response, for example, instruction cache miss. As a result, the fetch unit does not benefit from non-blocking instruction caches when instruction cache hit latency is more than one cycle.

Similarly, most of the benchmarks that show high inaccuracy on Multi2Sim, for example *gcc*_200, *hmmer*, *h*264*ref*, *gemsFDTD*, *namd*, and *perlbench*. also show high branch predictor misses. These misses have a significant impact on the overall performance results. Like gem5, many of these benchmarks contain a high number of branch operations (20% or more of the overall instruction mix). Other simulators also show exaggerated numbers of branch predictor and cache misses in the case of higher IPC inaccuracies for example: *h*264*ref*, *libquantum*, *milc* and *povray* on MARSSx86 and *libquantum* and *mcf* on ZSim.

It is noteworthy that the underestimated IPC values are not always explained by the aforementioned overestimated branch mispredictions. For instance, although Sniper shows overestimated branch mispredictions for *gromacs*, the actual number of these mispredictions is low and does not have a significant impact on performance results. The accumulation of the inaccuracy of upper levels of cache misses into lower-level cache misses explains higher inaccuracy in the number of last-level cache misses. Moreover, the inability to simulate μ-op fusion and μ-op cache (used in Haswell) in the studied simulators can cause more inaccuracies. Lack of other micro-architectural details in simulation models and flexible reconfiguration options in addition to the abstraction level of some simulators can also produce errors in the simulated results.

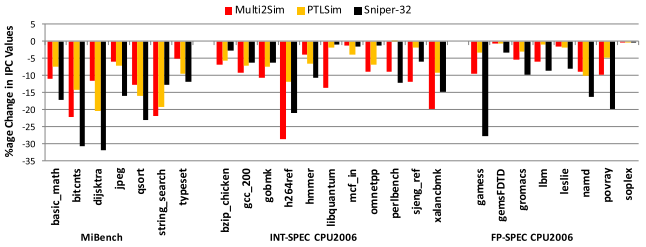

FIGURE 13. Percentage change in IPC values for reduced pipeline width for 32-bit binaries.

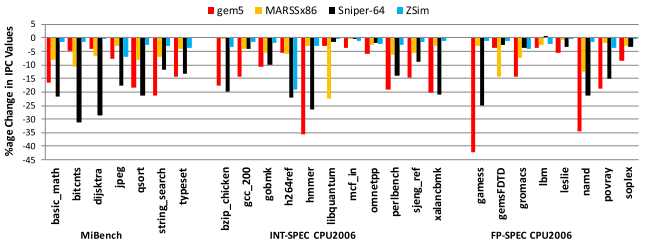

FIGURE 14. Percentage change in IPC values for reduced pipeline width for 64-bit binaries.

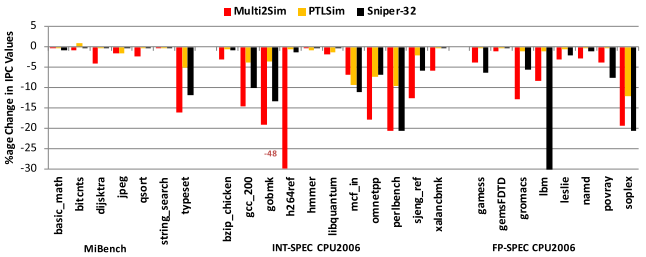

FIGURE 15. Percentage change in IPC values for reduced cache size for 32-bit binaries.

B. SENSITIVITY TESTS

Several existing studies rely on relative performance improvements informed by simulators to design and compare new architectures and ideas. To study and compare the relative performance of simulators, we performed three relative performance tests: (1) changing the width of pipeline stages to half of their normal values, (2) reducing the size of all caches to half of their sizes from Table 5 and (3) configuring a bimodal branch predictor instead of branch predictor used for Haswell. Figures 13 - 18 show the changes in IPC values (relative to the simulated Haswell target IPC) for these runs. It is hard to judge what the impact on IPC should be relative to the base configurations, without additional experiments, however; few observations can be made. For example, memory intensive benchmarks (like *mcf*, *gemsFDTD*, *xalancbmk*) are expected to show a bigger change in relative performance for reduced cache size compared to the other benchmarks. MARSSx86 and Sniper seem to be the most sensitive simulators for cache size change. However, MARSSx86 shows inconsistent behavior for cache size change for other benchmarks (examples *sjeng_ref* and *namd*), compared to the other simulators. In general, Sniper and gem5 seem to be more sensitive than other simulators to most of the changes in most of the cases, depending on benchmarks' sensitivity to the mentioned change. Zsim seems to be less affected by the change of the pipeline width amongst all the simulators. Although it is hard to asses the relative performance of the simulators and judge their relative accuracy, the figures clearly show that the relative performance of the simulators differs from each other and that the difference can be significant for some cases.

In addition to single-core experimental error and relative performance experiments, we compared the simulators for multicore experimental errors. Figures 19 and 20 show normalized IPC values for dual core and quad core runs for gem5, MARSSx86, Sniper (64 bit binaries) and ZSim, to the actual hardware results. We used multiprogrammed workload from SPEC CPU2006 benchmarks as inputs to the multicore simulations. Using multithreaded benchmarks is also interesting, however; it is not straightforward and it is time consuming to set up those benchmarks for all simulators. Our dual-core and quad-core input combinations were selected from CPU2006 benchmarks using a random number generator. The

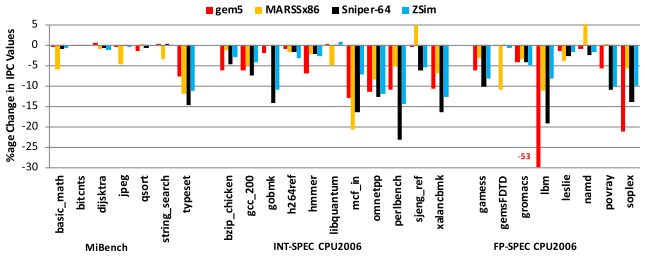

FIGURE 16. Percentage change in IPC values for reduced cache size for 64-bit binaries.

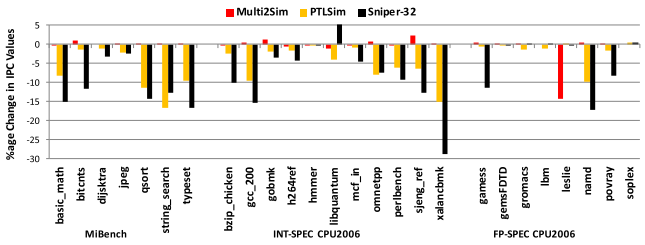

FIGURE 17. Percentage change in IPC values for change in branch predictor for 32-bit binaries.

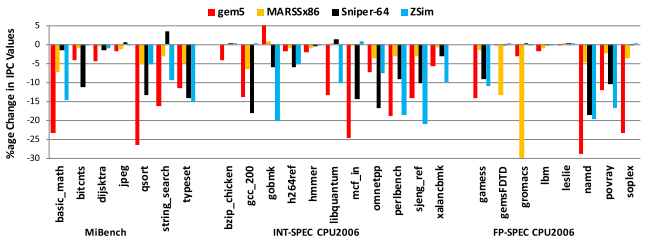

FIGURE 18. Percentage change in IPC values for change in branch predictor for 64-bit binaries.

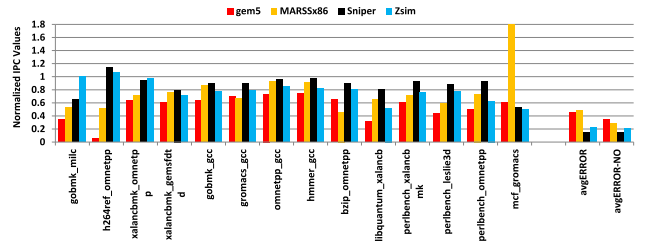

FIGURE 19. Normalized IPC for dual core runs.

mean average error rate is higher for multicore runs than for single-core runs. Sniper shows the highest accuracy for the dual core runs and ZSim shows the highest accuracy for the quad-core runs.

C. SIMULATION SPEED

Finally, we measured the speed of each simulator and the time it takes to fast-forward simulations. Table 7 shows the average simulation time for the simulators for each type of benchmarks. ZSim is the fastest simulator out of all the studied simulators. The table also shows the fast forwarding time by all simulators when simulating SPEC CPU benchmarks.

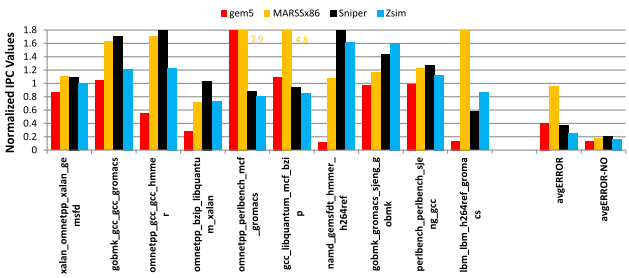

FIGURE 20. Normalized IPC for quad core runs.

TABLE 7. Average simulation time in seconds.

Benchmarks	Simulator	Fast Forwarding	Detailed Simulation
MiBench	gem5	NA	8576.68
	MARSSx86	NA	3250.16
	Sniper	NA	1001.91
	ZSim	NA	44.05
INT-SPEC2006	gem5	338736	342724.15
	MARSSx86	3794	8853
	Sniper	2892	4052
	ZSim	2027	2172
FP-SPEC2006	gem5	112851	139080
	MARSSx86	4182	6183
	Sniper	723	2160
	ZSim	596	801
Note: MiBench benchmarks are simulated completely without any fast-forwarding			

D. SUMMARY OF OBSERVATIONS

Below is a summary of our main observations based on our survey of the different simulators and the experimental error study.

- Experiments point out a strong correlation of a simulator's accuracy, compared to real hardware, and the existence of a validation and a calibration of simulators for the corresponding target architecture. For instance, Sniper and ZSim, which show least error in our experiments, have been validated and calibrated for Intel Nehalem and Westmere cores respectively [23], [143].
- The results of uncalibrated/invalidated simulators can diverge from the reference hardware performance numbers significantly. This is also observed by Asri et al. when calibrated MARSSx86 for a particular target machine [140].
- Highly inaccurate branch predictor and cache misses, inaccurate instruction to μ-ops decoding, and the absence of some Haswell optimization structures in the studied simulators are the main causes of inaccuracies in the simulation results.
- The relative accuracy of simulators can vary significantly, which can result in inaccurate conclusions.
- A more accurate simulator may still not be able to fit your needs. For instance, Sniper and ZSim are not very flexible for modeling new micro-architectural features compared to gem5 and MARSSx86, although they

show greater accuracy. On the other hand, gem5 and MARSSx86 exhibit more flexibility, which aids testing new microarchitectural design ideas and studying the performance of particular micro-architectural blocks.

- Sniper and ZSim are compelling choices to simulate many-core x86 architectures because they are faster and produce better accuracy than the other simulators.
- gem5 and MARSSx86 support full-system simulations and can be used to simulate applications with system calls, for better accuracy when applications contain many system calls. They can also be used to study the effect of OS interaction with hardware on the performance of applications. ZSim and Sniper are application-level simulators.
- MARSSx86's reasonable accuracy, detailed simulation model, full system support and good speed makes it a good option for detailed full system studies especially for multicore targets. PTLsim can act as a foundation to build more advanced simulators, for example MARSSx86 [4].
- An important use case of Multi2Sim is CPU-GPU architecture simulation.
- gem5's configurability, support of various ISAs, and support of a complete OS in addition to its active community of developers, makes it a convincing choice for running comprehensive experiments on a particular processor block or an entire core, to study hardware and OS interaction or to study heterogeneous systems (single-ISA, multi-ISA, CPU-GPU, CPU-ASIC).

X. CONCLUSIONS AND FUTURE DIRECTIONS

This paper presents a detailed study of computer architectural simulators. We performed a comparative survey of various simulators and classified them into different categories. We tested the absolute and relative accuracies of six contemporary computer architecture simulators by comparing their simulation results to those of a real hardware. The experimental results show that Sniper produces the minimum absolute error, and that ZSim is the fastest for single-core simulations. However, picking a simulator to use depends on the purpose of the study/research and the features of the simulator. This work also stresses on the importance of validating simulators and points out some sources of inaccuracies in computer architecture simulators.

With the emergence of many-core architectures, new applications, and heterogenous design options, there is a greater need for new and innovative simulation acceleration techniques without sacrificing the accuracy of simulators. In addition, creating modular simulators and having up-to-date full documentations make simulators more flexible and easier to use. The integration of more heterogeneous components within processor architecture simulators enables new research studies and directions, especially when the interaction between processor cores and the heterogeneous components and accelerators is flexibly modeled. Moreover, as Moore's law is coming to an end, future architectural innovations will require optimizations across the entire software/hardware computing stack. Current architectural simulators were not designed to study such cross-layered optimizations. As such, there is an urgent need to come up with new simulation techniques that will allow researchers to experiment their ideas across the entire stack.

Finally, simulation validation is very important to ensure confidence in the produced results and facilitate reproducibility. Although many studies rely on the relative performance of simulators, our experiments showed that different simulators show different relative performance and that the difference can be significant. Thus, there is a need for new methods to validate the relative accuracy of simulators. Similarly, there is a need to validate simulators for multicore processor simulation, as multicore simulation inaccuracies for simulators that were validated for single core processors show high error rates for multicore experiments.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable feedback and comments.

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, no. 8, pp. 30–36, Aug. 2003.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, T. Krishna, T. Krishna, T. Krishna, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The GEM5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[3] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2Sim: A simulation framework to evaluate multicore-multithreaded processors," in *Proc. 19th Int. Symp. Comput. Archit. High Perform. Comput.*, Rio Grande do Sul, Brazil, Oct. 2007, pp. 62–68.

[4] A. Patel, F. Afram, and K. Ghose, "MARSS-x86: A qemu-based microarchitectural and systems simulator for x86 multicore processors," in *Proc. 1st Int. QEMU Users Forum*, Grenoble, France, Mar. 2011, pp. 29–30.

[5] M. T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, San Jose, CA, USA, Apr. 2007, pp. 23–34.

[6] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Seattle, WA, USA, Nov. 2011, pp. 1–12.

[7] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, Jun. 2013, pp. 475–486.

[8] W. Yurcik, G. S. Wolffe, and M. A. Holliday, "A survey of simulators used in computer organization/architecture courses," in *Proc. Summer Comput. Simulation Conf.*, 2001, pp. 524–529.

[9] P. Prasad, A. Alasdoon, A. Beg, and A. Chan, "Simulators for teaching computer organization and architecture—A brief survey," in *Proc. Int. Conf. Edu. Inf. Manage.*, 2013, pp. 14–18.

[10] 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 Trans. Educ.*, vol. 52, no. 4, pp. 449–458, Nov. 2009.

[11] R. Hasan and S. Mahmood, "Survey and evaluation of simulators suitable for teaching for computer architecture and organization supporting undergraduate students at Sir Syed University of Engineering & Technology," in *Proc. UKACC Int. Conf. Control*, Sep. 2012, pp. 1043–1045.

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

[13] M. A. Holliday, "Techniques for cache and memory simulation using address reference traces," *Int. J. Comput. Simul.*, vol. 1, no. 1, pp. 129–151, 1990.

[14] U. Urden, "A comparison of three computer system simulators," M.S. thesis, Dept. Syst. Softw. Eng., Blekinge Inst. Technol., Karlskrona, Sweden, 2004.
[15] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, no. 6, pp. 4–12, Nov./Dec. 2015.
[16] T. M. Aamodt, W. Fung, I. Singh, A. El-Shafiey, J. Kwa, T. Hetherington, A. Gubran, A. Boktor, T. Rogers, and A. Bakhoda, "GPGPU-Sim3.x manual," Dept. Elect. Comput. Eng., Univ. Brit. Columbia, Vancouver, BC, Canada, 2012.
[17] 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 *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Dec. 2009, pp. 469–480.
[18] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487–498, 2013.
[19] R. Desikan, D. Burger, and S. W. Keckler, "Measuring experimental error in microprocessor simulation," in *Proc. 28th Annu. Int. Symp. Comput. Archit.*, Gothenburg, Sweden, Jun./Jul. 2001, pp. 266–277.
[20] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proc. IEEE Int. Symp. Comput. Archit. (ISCA)*, San Diego, CA, USA, Jun. 2003, pp. 84–95.
[21] D. G. Perez, G. Mouchard, and O. Temam, "MicroLib: A case for the quantitative comparison of micro-architecture mechanisms," in *Proc. 37th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Portland, OR, USA, Dec. 2004, pp. 43–54.
[22] 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 *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Monterey, CA, USA, Mar. 2014, pp. 13–22.
[23] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 3, p. 28, 2014.
[24] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux, "SiNUCA: A validated micro-architecture simulator," in *Proc. IEEE High Perform. Comput. Commun. (HPCC)*, New York, NY, USA, Aug. 2015, pp. 605–610.
[25] Y. Kim, W. Yan, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, Jan. 2016.
[26] M. Dubois, M. Annavaram, and P. Stenström, *Parallel Computer Organization and Design*. Cambridge, U.K.: Cambridge Univ. Press, 2012, ch. 9, pp. 490–504.
[27] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.
[28] D. Aarno and J. Engblom, *Software and System Development Using Virtual Platforms: Full-System Simulation With Wind River Simics*. San Mateo, CA, USA: Morgan Kaufmann, 2014.
[29] K. Kise, T. Katagiri, H. Honda, and T. Yuba, "The simcore/alpha functional simulator," in *Proc. Workshop Comput. Archit. Educ., Held Conjunction 31st Int. Symp. Comput. Archit.*, Munich, Germany, Jun. 2004, p. 24.
[30] D. Patti, A. Spadaccini, M. Palesi, F. Fazzino, and V. Catania, "Supporting undergraduate computer architecture students using a visual MIPS64 CPU simulator," *IEEE Trans. Educ.*, vol. 55, no. 3, pp. 406–411, Aug. 2012.
[31] R. N. Ibbett, P. E. Heywood, and F. W. Howell, "HASE: A flexible toolset for computer architects," *Comput. J.*, vol. 38, no. 10, pp. 755–764, 1995.
[32] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A parallel functional simulator for GPGPU," in *Proc. IEEE Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Miami Beach, FL, USA, Aug. 2010, pp. 351–360.
[33] L. Eeckhout, *Computer Architecture Performance Evaluation Methods*. San Rafael, CA, USA: Morgan & Claypool, 2010.
[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 *Proc. Workshop Comput. Archit. Educ., Held Conjunction 31st Int. Symp. Comput. Archit.*, Munich, Germany, Jun. 2004, pp. 22–30.
[35] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob, "CMP$im: A pin-based on-the-fly multi-core cache simulator," in *Proc. 4th Annu. Workshop Modeling, Benchmarking Simulation (MoBS), co-Located ISCA*, Beijing, China, 2008, pp. 28–36.
[36] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2007, pp. 250–261.
[37] J. Sharkey, D. Ponomarev, and K. Ghose, "M-SIM: A flexible, multithreaded architectural simulation environment," Dept. Comput. Sci., State Univ. New York Binghamton, Binghamton, NY, USA, Tech. Rep. CS-TR-05-DP01, Oct. 2005.
[38] M. M. K. 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," *ACM SIGARCH Comput. Archit. News Lett.*, vol. 33, no. 4, pp. 92–99, 2005.
[39] M. Reilly and J. Edmondson, "Performance simulation of an Alpha microprocessor," *Computer*, vol. 31, no. 5, pp. 50–58, May 1998.
[40] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. (Aug. 5, 2005). *SESC Simulator*. [Online]. Available: http://sesc.sourceforge.net/
[41] J. E. Veenstra and R. J. Fowler, "MINT: A front end for efficient simulation of shared-memory multiprocessors," in *Proc. Int. Workshop Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Durham, NC, USA, Jan./Feb. 1994, pp. 201–207.
[42] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: Simulating shared-memory multiprocessors with ILP processors," *Computer*, vol. 35, no. 2, pp. 40–49, Feb. 2002.
[43] (2015). *Flexus*. Accessed: Aug. 6. 2015. [Online]. Available: http://parsa.epfl.ch/simflex/
[44] 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 Perform. Eval. Rev.*, vol. 31, no. 4, pp. 31–34, Mar. 2004.
[45] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *Proc. 16th IEEE Int. Symp. High Perform. Comput. Archit.*, Bangalore, India, Jan. 2010, pp. 1–12.
[46] 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, no. 4, pp. 52–60, Jul./Aug. 2006.
[47] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, Jan. 2010, pp. 1–12.
[48] Z. Dong, J. Wang, G. Riley, and S. Yalamanchili, "An efficient front-end for timing-directed parallel simulation of multi-core system," in *Proc. 7th Int. Conf. Simulation Tools Techn. (ICST)* SIMUTools. Brussels, Belgium: ICST, 2014, pp. 201–206. doi: 10.4108/icst.simutools.2014.254638.
[49] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, "Asim: A performance model framework," *Computer*, vol. 35, no. 2, pp. 68–76, Feb. 2002.
[50] C. J. Mauer, M. D. Hill, and D. A. Wood, "Full-system timing-first simulation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 108–116, Jun. 2002.
[51] J. Chen, M. Dubois, and P. Stenstrom, "Integrating complete-system and user-level performance/power simulators: The SimWattch approach," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Austin, TX, USA, Mar. 2003, pp. 1–10.
[52] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. 27th Int. Symp. Comput. Archit.*, Vancouver, BC, Canada, Jun. 2000, pp. 83–94.
[53] N. Neelakantam, C. Blundell, J. Devietti, M. M. Martin, and C. Zilles, "FeS2: A full-system execution-driven simulator for x86," *Poster Presented ASPLOS*, vol. 2008, p. 6, Mar. 2008.
[54] S. A. Herrod, "Using complete machine simulation to understand computer system behavior," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 1998.

[55] P. Stanley-Marbell and D. Marculescu, "*Sunflower*: Full-system, embedded microarchitecture evaluation," in *High Performance Embedded Architectures and Compilers*. Berlin, Germany: Springer, 2007, pp. 168–182.

[56] L. Schaelicke and M. Parker, "The design and utility of the ML-RSIM system simulator," *J. Syst. Archit.*, vol. 52, no. 5, pp. 283–297, 2006.

[57] L. K. John, "Performance evaluation: Techniques, tools, and benchmarks," *Comput. Eng. Handbook*, vol. 8, no. 1, p. 21, Dec. 2001.

[58] P. Crowley and J.-L. Baer, "On the use of trace sampling for architectural studies of desktop applications," in *Proc. Workload Characterization, Methodol. Case Stud., Based 1st Workshop Workload Characterization*, Dallas, TX, USA, 1999, pp. 15–24.

[59] R. Bhargava, L. K. John, and F. Matus, "Accurately modeling speculative instruction fetching in trace-driven simulation," in *Proc. IEEE Int. Perform., Comput. Commun. Conf.*, Scottsdale, AZ, USA, Feb. 1999, pp. 65–71.

[60] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Austin, TX, USA, Apr. 2013, pp. 74–85.

[61] S. R. Goldschmidt and J. L. Hennessy, "The accuracy of trace-driven simulations of multiprocessors," in *Proc. ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, Santa Clara, CA, USA, May 1993, pp. 146–157.

[62] R. F. Cmelik and D. Keppel, "Shade: A fast instruction-set simulator for execution profiling," Sun Microsyst., Mountain View, CA, USA, Tech. Rep. UWCSE 93-06-06, 1993.

[63] R. A. Sugumar and S. G. Abraham, "Efficient simulation of caches under OPT replacement with applications to miss characterization," in *Proc. ACM SIGMETRICS Conf.*, 1993.

[64] E. Larson, S. Chatterjee, and T. Austin, "MASE: A novel infrastructure for detailed microarchitectural modeling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Tucson, AZ, USA, Nov. 2001, pp. 1–9.

[65] M. E. Elrabaa, A. Hroub, M. F. Mudawar, A. Al-Aghbari, M. Al-Asli, and A. Khayyat, "A very fast trace-driven simulation platform for chip-multiprocessors architectural explorations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3033–3045, Nov. 2017.

[66] K. Sangaiah, M. Lui, R. Jagtap, S. Diestelhorst, S. Nilakantan, A. More, B. Taskin, and M. Hempstead, "Synchrotrace: Synchronization-aware architecture-agnostic traces for lightweight multicore simulation of CMP and HPC workloads," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 1, p. 2, 2018.

[67] J. R. Jump. (Sep. 3, 2015). *YACSIM Reference Manual*. [Online]. Available: http://www.ece.rice.edu/~rsim/pubs/yacsim.man.ps

[68] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, Shenzhen, China, Feb. 2013, pp. 448–459.

[69] H. Davis, S. R. Goldschmidt, and J. L. Hennessy, "Tango: A multiprocessor simulation and tracing system," Dept. Comput. Syst. Lab., Stanford Univ., Stanford, CA, USA, Tech. Rep. CSL-TR-90-439, Jul. 1990.

[70] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl, "Proteus: A high-performance parallel-architecture simulator," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. MIT/LCS/TR-516, Sep. 1991.

[71] J. Huang, "The simulator for multithreaded computer architecture," Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep. ARCTiC 00-05, 2000.

[72] A. Sharma, A.-T. Nguyen, J. Torellas, M. Michael, and J. Carbajal, "Augmint: A multiprocessor simulation environment for Intel x86 architectures," Univ. Illinois Urbana-Champaign, Urbana, IL, USA, Tech. Rep. 1463, Mar. 1996.

[73] S. A. Herrod. (Aug. 5, 2015). *Tango Lite: A Multiprocessor Simulation Environment, Introduction and User's Guide*. [Online]. Available: http://www-flash.stanford.edu/herrod/docs/tango-lite.ps

[74] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," Compaq Comput. Corp., Palo Alto, CA, USA, Tech. Rep. 2001/2, Aug. 2001.

[75] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM J. Res. Develop.*, vol. 47, nos. 5–6, pp. 653–670, Sep. 2003.

[76] T. M. Austin, T. N. Mudge, and D. Grunwald. (Sep. 3, 2015). *PowerAnalyzer for Pocket Computers*. [Online]. Available: http://www.researchgate.net/profile/Todd_Austin2/publication/265357531_PowerAnalyzer_for_Pocket_Computers/links/54bf0ba40cf28ce68e6b0cb1.pdf

[77] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: A cycle-accurate energy estimation tool," in *Proc. 37th Annu. Design Autom. Conf.*, Los Angeles, CA, USA, Jun. 2000, pp. 340–345.

[78] J. Nayfach-Battilana and J. Renau, "SOI, interconnect, package, and mainboard thermal characterization," in *Proc. 14th ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, San Francisco, CA, USA, Aug. 2009, pp. 327–330.

[79] K. Skadron, M. Stan, M. Barcella, A. Dwarka, W. Huang, Y. Li, Y. Ma, A. Naidu, D. Parikh, and P. Re, "HotSpot: Techniques for modeling thermal effects at the processor-architecture level," in *Proc. Int. Workshop Thermal Invest. ICs Syst.*, Madrid, Spain, Oct. 2002, pp. 1–4.

[80] A. Ziabari, J.-H. Park, E. K. Ardestani, J. Renau, S.-M. Kang, and A. Shakouri, "Power blurring: Fast static and transient thermal analysis method for packaged integrated circuits and power devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2366–2379, Nov. 2014.

[81] A. Ziabari, E. K. Ardestani, J. Renau, and A. Shakouri, "Fast thermal simulators for architecture level integrated circuit design," in *Proc. 27th Annu. IEEE Semiconductor Therm. Meas. Manage. Symp.*, San Jose, CA, USA, Mar. 2011, pp. 70–75.

[82] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.* New York, NY, USA: ACM, 2012, pp. 355–366.

[83] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: A memory system simulator," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 100–107, 2005.

[84] J. Edler and M. D. Hill. (Sep. 3, 2015). *Dinero IV Trace-Driven Uniprocessor Cache Simulator*. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/

[85] M. L. C. Cabeza, M. I. G. Clemente, and M. L. Rubio, "CacheSim: A cache simulator for teaching memory hierarchy behaviour," *ACM Special Interest Group Comput. Sci. Educ. Bull.*, vol. 31, no. 3, p. 181, Sep. 1999.

[86] A. Faravelon, N. Fournel, and F. Pétrot, "Fast and accurate branch predictor simulation," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.* San Jose, CA, USA: EDA Consortium, 2015, pp. 317–320.

[87] *Championship Value Prediction*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.microarch.org/cvp1/

[88] T. Cortes and J. Labarta, "Hraid: A flexible storage-system simulator," in *Proc. PDPTA*, 1999, pp. 772–778.

[89] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "Flashsim: A simulator for nand flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simulation (SIMUL)*, Sep. 2009, pp. 125–131.

[90] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, Oakland, CA, USA, 2018, pp. 49–66.

[91] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2009, pp. 33–42.

[92] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 86–96.

[93] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proc. IEEE 26th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2015, pp. 162–163.

[94] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Boston, MA, USA, Apr. 2009, pp. 163–174.

[95] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2014, pp. 97–108.

[96] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 380–387.

[97] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, and J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 267–278, 2016.
[98] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 29–42.
[99] ALDEC Company. *ActiveHDL Simulator*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.aldec.com/en/products/fpga_simulation/active-hdl
[100] Xilinx Incorporate. *ISE Simulator (ISim)*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.xilinx.com/products/design-tools/isim.html
[101] *Cadence Design Systems Incroporate. Incisive Enterprise Simulator*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/incisive-enterprise-simulator.html
[102] *Mentor Graphics*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.mentor.com/
[103] Intel Corporation. *Quartus II Simulator (QSIM)*. Accessed: Jan. 1, 2019. [Online]. Available: https://www.intel.com/
[104] V. Cuppu, *Cycle Accurate Simulator for TMS320C62x, 8 Way VLIW DSP Processor*. College Park, MD, USA: Univ. Maryland, 1999.
[105] V. Zivojnovic, C. Schlager, and J. Fitzner, "System-level modeling of DSP and embedded processors," in *Proc. Conf. Rec. 22nd Asilomar Conf. Signals, Syst. Comput.*, vol. 2, Nov. 1998, pp. 1730–1734.
[106] N. Sedaghati-Mokhtari, M. Nazm-Bojnordi, A. Hormati, and S. M. Fakhraie, "An efficient and extendable modeling approach for VLIW DSP processors," in *Advances in Computer Science and Engineering*. Berlin, Germany: Springer, 2008, pp. 267–274.
[107] M. Vachharajani, N. Vachharajani, D. Penry, J. Blome, and D. August, "Microarchitectural exploration with Liberty," in *Proc. 35th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Istanbul, Turkey, Nov. 2002, pp. 271–282.
[108] N. Freeman, "Soonergy: A pluggable, cycle-accurate computer architecture simulator," M.S. thesis, Dept. Elect. Comput. Eng., Univ. Oklahoma, Norman, OK, USA, May 2011.
[109] L. Sawalha, S. Wolff, M. P. Tull, and R. D. Barnes, "Phase-guided scheduling on single-ISA heterogeneous multicore processors," in *Proc. 14th Euromicro Conf. Digit. Syst. Design*, Aug./Sep. 2011, pp. 736–745.
[110] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTson: Infrastructure for full system simulation," *SIGOPS Opererating Syst. Rev.*, vol. 43, no. 1, pp. 52–61, 2009.
[111] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, San Antonio, TX, USA, Feb. 2011, pp. 406–417.
[112] S. Hassani, G. Southern, and J. Renau, "Livesim: Going live with microarchitecture simulation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 606–617.
[113] S. Simeonov and G. M. Schneider, "Msim: An improved microcode simulator," *ACM Special Interest Group Comput. Sci. Educ. Bull.*, vol. 27, no. 2, pp. 13–17, Jun. 1995.
[114] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete computer system simulation: The SimOS approach," *IEEE Parallel Distrib. Technol., Syst. Appl.*, vol. 3, no. 4, pp. 34–43, Mar. 1995.
[115] D. M. Tullsen, "Simulation and modeling of a simultaneous multithreading processor," in *Proc. 22nd Annu. Comput. Meas. Group Conf.*, 1996, pp. 819–828.
[116] J. Larus. (Jul. 1, 2016). *Spim: A Mips32 Simulator*. [Online]. Available: http://pages.cs.wisc.edu/~larus/spim.html
[117] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch, "TEM^{2}P^{2}EST: A thermal enabled multi-model power/performance ESTimator," in *Proc. Int. Workshop Power-Aware Comput. Syst.*, Cambridge, MA, USA, 2000, pp. 112–125.
[118] M. Moudgill, "Techniques for implementing fast processor simulators," in *Proc. 31st Annu. Simulation Symp.*, Apr. 1998, pp. 83–90.
[119] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, M. D. Hill, D. A. Wood, S. Huss-Lederman, and J. R. Larus, "Wisconsin Wind Tunnel II: A fast, portable parallel architecture simulator," *IEEE Concurrency*, vol. 8, no. 4, pp. 12–20, Oct. 2000.
[120] G. H. Loh, S. Subramaniam, and Y. Xie, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Software.*, Boston, MA, USA, Apr. 2009, pp. 53–64.
[121] J. J. Yi and D. J. Lilja, "Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations," *IEEE Trans. Comput.*, vol. 55, no. 3, pp. 268–280, Mar. 2006.
[122] A. KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja, "Adapting the SPEC 2000 benchmark suite for simulation-based computer architecture research," in *Workload Characterization of Emerging Computer Applications*. Boston, MA, USA: Springer, 2000, pp. 83–100.
[123] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.
[124] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?" in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 271–282.
[125] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 318–319, Jun. 2003.
[126] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. 10th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Oct. 2002, pp. 45–57.
[127] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins, "Characterizing and comparing prevailing simulation techniques," in *Proc. 11th Int. Symp. High-Perform. Comput. Archit.*, San Francisco, CA, USA, Feb. 2005, pp. 266–277.
[128] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, Jul. 2006.
[129] B. Randell, P. Lee, and P. C. Treleaven, "Reliability issues in computing system design," *ACM Comput. Surv.*, vol. 10, no. 2, pp. 123–165, 1978.
[130] M. T. Cruz, S. Bischoff, and R. Rusitoru, "Shifting the barrier: Extending the boundaries of the BarrierPoint methodology," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Belfast, U.K., Apr. 2018, pp. 120–122. doi: 10.1109/ISPASS.2018.00023.
[131] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. D. Bosschere, "Statistical simulation: Adding efficiency to the computer designer's toolbox," *IEEE Micro*, vol. 23, no. 5, pp. 26–38, Sep. 2003.
[132] S. Nussbaum and J. E. Smith, "Modeling superscalar processors via statistical simulation," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Barcelona, Spain, Sep. 2001, pp. 15–24.
[133] L. Ricciulli, P. Lincoln, and J. Meseguer, *Execution-Driven Distributed Simulation of Parallel Architectures*. Menlo Park, CA, USA: SRI International, 1995.
[134] G. Zheng, G. Kakulapati, and L. V. Kale, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Santa Fe, NM, USA, Apr. 2004, pp. 78–88.
[135] (2015). *RISC-V Foundation*. [Online]. Available: https://riscv.org/
[136] N. Choudhary, S. Wadhavkar, T. Shah, H. Mayukh, J. Gandhi, B. Dwiel, S. Navada, H. Najaf-Abadi, and E. Rotenberg, "FabScalar: Automating superscalar core design," *IEEE Micro*, vol. 32, no. 3, pp. 48–59, May/Jun. 2012.
[137] B. Black and J. P. Shen, "Calibration of microprocessor performance models," *Computer*, vol. 31, no. 5, pp. 59–65, May 1998.
[138] H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H. Lipasti, "Precise and accurate processor simulation," in *Proc. Workshop Comput. Archit. Eval. Using Commercial Workloads, 8th Int. Symp. High-Perform. Comput. Archit.*, vol. 8. Cambridge, MA, USA, Feb. 2002, pp. 1–10.
[139] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich, "FLASH vs. (simulated) FLASH: Closing the simulation loop," *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 5, Dec. 2000, pp. 49–58.
[140] M. Asri, A. Pedram, L. K. John, and A. Gerstlauer, "Simulator calibration for accelerator-rich architecture studies," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling Simulation*, Samos, Greece, Jul. 2016, pp. 88–95.
[141] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of GEM5 simulator system," in *Proc. 7th Int. Workshop Reconfigurable Commun.-Centric Syst.-Chip*, York, U.K., Jul. 2012, pp. 1–7.
[142] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.*, Minneapolis, MN, USA, 2012, pp. 335–344.

[143] ZSim. (Jul. 1, 2016). *ZSim Tutorial Validation*. [Online]. Available: http://zsim.csail.mit.edu/tutorial/slides/validation.pdf
[144] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Hardware-validated CPU performance and energy modelling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Belfast, U.K., Apr. 2018, pp. 44–53.
[145] R. Desikan, D. Burger, S. W. Keckler, and T. Austin, "Sim-alpha: A validated, execution-driven alpha 21264 simulator," Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, USA, Tech. Rep. TR-01-23, 2001.
[146] SPEC. (Aug. 5, 2015). *SPEC CPU*. [Online]. Available: https://www.spec.org/cpu2000/
[147] Perf. (Aug. 5, 2015). *Perf: Linux Profiling With Performance Counters*. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page
[148] A. Akram and L. Sawalha, "x86 computer architecture simulators: A comparative study," in *Proc. IEEE Int. Conf. Comput. Design*, Pheonix, AZ, USA, Oct. 2016, pp. 638–645.
[149] J.-E. Jo, G.-H. Lee, H. Jang, J. Lee, M. Ajdari, and J. Kim, "Diagsim: Systematically diagnosing simulators for healthy simulations," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 1, p. 4, 2018.
[150] T. Tanimoto, T. Ono, and K. Inoue, "Dependence graph model for accurate critical path analysis on out-of-order processors," *J. Inf. Process.*, vol. 25, pp. 983–992, 2017.
[151] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. USENIX Annu. Tech. Conf., FREENIX Track*, Anaheim, CA, USA, Apr. 2005, pp. 41–46.
[152] AMD. (Aug. 5, 2015). *APP SDK—A Complete Development Platform*. [Online]. Available: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/
[153] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. 19th ACM Symp. Operating Syst. Princ.*, Bolton Landing, NY, USA, 2003, vol. 37, no. 5, pp. 164–177.
[154] A. Tridgell and P. Mackerras, "The rsync algorithm," Austral. Nat. Univ., Canberra, ACT, Australia, Tech. Rep. TR-CS-96-05, Jun. 1996.
[155] N. B. Mallya, C. Gonzalez-Alvarez, and T. E. Carlson, "Flexible timing simulation of RISC-V processors with sniper," in *Proc. 2nd Workshop Comput. Archit. Res. RISC-V (CARRV)*, Los Angeles, CA, USA, Jun. 2018.
[156] D. Sanchez and C. Kozyrakis, "The ZCache: Decoupling ways and associativity," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 187–198.
[157] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "GEM5-GPU: A heterogeneous CPU-GPU simulator," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 34–36, Jan./Jun. 2015.
[158] IntelManuals. (Aug. 5, 2015). *Intel 64 and IA-32 Architectures Software Developer Manuals*. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
[159] A. Fog. (Sep. 3, 2015). *Instruction Tables: Lists of Instruction Latencies, Throughputs and Micro-Operation Breakdowns for Intel, AMD and VIA CPUs*. [Online]. Available: http://www.agner.org/optimize/instruction_tables.pdf
[160] D. Kanter. (Aug. 5, 2015). *Intel's Haswell CPU Micrarchitecture*. [Online]. Available: http://www.realworldtech.com/haswell-cpu/
[161] A. L. Shimpi. (Aug. 5, 2015). *Intel's Haswell Architecture Analyzed: Building a New PC and a New Intel*. [Online]. Available: http://www.anandtech.com/show/6355/intels-haswell-architecture/6
[162] A. Akram and L. Sawalha, "A comparison of x86 computer architecture simulators," Western Michigan Univ., Kalamazoo, MI, USA, Tech. Rep. TR-CASRL-1-2016, Oct. 2016.
[163] R. E. Kessler, E. J. McLellan, and D. A. Webb, "The Alpha 21264 microprocessor architecture," in *Proc. Int. Conf. Comput. Design, VLSI Comput. Processors*, Austin, TX, USA, Oct. 1998, pp. 90–95.
[164] S. McFarling, "Branch predictor with serially connected predictor stages for improving branch prediction accuracy," U.S. Patent 6 374 349 B2, Apr. 16, 2002.
[165] V. Uzelac and A. Milenkovic, "Experiment flows and microbenchmarks for reverse engineering of branch predictor structures," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Boston, MA, USA, Apr. 2009, pp. 207–217.
[166] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero, "Vector extensions for decision support dbms acceleration," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Vancouver, BC, Canada, Dec. 2012, pp. 166–176.
[167] SPEC. (Aug. 5, 2015). *SPEC CPU*. [Online]. Available: https://www.spec.org/cpu2006/
[168] 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 *Proc. IEEE 4th Annu. Workshop Workload Characterization, Held Conjunct 34th Annu. IEEE/ACM Symp. Microarchitecture*, Austin, TX, USA, Dec. 2001, pp. 3–14.
[169] PAPI. (Aug. 5, 2015). *Performance Application Programming Interface*. [Online]. Available: http://icl.cs.utk.edu/papi/
[170] S. Diestelhorst. (Aug. 5, 2015). *PTLsim Source Code*. [Online]. Available: https://github.com/stephand/ptlsim

AYAZ AKRAM received the bachelor's degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and the master's degree in computer engineering from Western Michigan University. He is currently pursuing the Ph.D. degree in computer science with the University of California at Davis, Davis. His research interests include computer architecture, high-performance computing, and the intersection of computer architecture with other areas like computer security and machine learning.

LINA SAWALHA received the bachelor's degree in computer engineering from the Jordan University of Science and Technology, and the master's and Ph.D. degrees in electrical and computer engineering from The University of Oklahoma, in 2009 and 2012, respectively. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Western Michigan University. Her research interests include computer architecture, heterogeneous architectures and systems, hardware/software codesign, and high-performance and energy-efficient computing. She is a Senior Member of ACM.