An Overview of Chip Multi-Processors Simulators Technology

Malik Al-Manasia[1] and Zenon Chaczko[2]

[1,2] Faculty of Engineering, University of Technology Sydney, Broadway, NSW 2007
malik.a.al-Manasia@student.uts.edu.au, zenon@eng.uts.edu.au

Abstract. Computer System Architecture (CSA) simulators are generally used to develop and validate new CSA designs and developments. The goal of this paper is to provide an insight into the importance of CSA simulation and the possible criteria that differentiate between various CSA simulators. Multi-dimensional aspects determine the taxonomy of CSA simulators including their accuracy, performance, functionality and flexibility. The *Sniper* simulator has been selected for a closer look and testing. The Sniper proofs its ability to scale to hundred cores with a wide range of functionality and performance.

Keywords: Simulators, Application Level, Simics, gem5, RSIM, SimOS, Graphite; Sniper; Zsim, McSimA+, CMPSim, benchmarks

1 Background

A Computer System Architecture (CSA) simulator is a software based tool, that is useful in modelling various components and devices of the computer to forecast the output and performance level given a particular input. Computer simulation is very often the driving force in computer architecture researches and designs. This is due to the enormous work in the design of compound microarchitecture, which is rarely modeled in an analytical manner making the process of prototyping and all design stages excessively expensive. A comprehensive simulation is equally expensive as it needs well crafted simulators, realistic workloads and numerous machine cycles.

A mandatory level of details is achieved at a slow pace, because even the high-speed simulators may require a period of many weeks or months to complete a task. This becomes even worse when multi-core processor designs become dominant. The total throughput of the host chip increases with increasing generations; however, the single-thread performance, which is a fundamental element in simulation, does not show any improvement because of limitations on the power. It is also necessary to note that a target-architecture model gradually becomes complex since there has to an increasing number of cores. The increased performance of the host cores does not adapt to the target increasing complexity.

Researchers are aware of the problem by exploring various strategies of reducing simulation time. Some earlier works have applied the sampling method [1, 2] to get a statistical representation of a given part of the application in order to provide similar

insights in an entire simulation, but in a short period. Other researchers [3-5] suggest conducting the parallel simulation in order to decrease a single-simulation period through operating it on numerous threads. This method is suitable for design phases, which needs a faster evaluation of one or more design points. However, if many parameters are considered, running multithreaded simulations simultaneously gives better simulation performance[6].

The strategies of reducing simulation period are necessary for cycle-accurate simulators [7, 8]. These simulators provide the fundamental elements of research in computer architecture to enhance understanding of the stresses relating to workload from the micro architectural structures. However, their time consuming characteristic is a limitation to their in the exploration of chips consisting of tens or more cores. Generally, simulators that are cycle accurate are not suitable for large-scale architectures[6].

2 CSA Simulators Importance

Simulations are vital and understanding of their role makes it possible to carry our various complex activities. Simulation can be important in a system migration as the old systems, which are expensive, can be simulated to keep them operational. Therefore, simulators can assist the organization in migrating to the new hardware and software easier. This allows for significant savings to business, in terms of finances and time. Simulation is also important in hardware development. According to Magnusson et. al, simulators can be useful in modelling computer systems regardless of their availability [9]. This makes the process of hardware development simpler since the hardware could be in the development process or still non-existent. In addition, it limit indirect costs as it reduces the need for costly hardware prototypes [10]. Simulation can be useful in saving costs. This is through simulating the costly hardware using a cheaper one. For instance, the standard x86-based PC, saving the organization, costs associated with expensive hardware. It is also useful in software development especially for developers of operating systems and device drivers. Programs operating in kernel mode can be very difficult to debug; however, simulators provide non-intrusive debugging options that make the development of such programs simpler. Simulators also function to provide security since they run in a controlled environment, which enhances security. This allows companies to be able to try the new systems before they are used in the process of production. This is important in boosting the company's overall security system. Simulators are useful in restoring previous states of simulated computers.

3 Taxonomy of CSA Simulators

Classifications of computer architecture simulators are categorized basing on their contexts. To start with, they are classified according to their **scope**, which include Microarchitecture, Full System, Application-based, Instruction Set and Cycle Accurate simulators. Microarchitecture simulation technique is useful in modeling the design and performance of microprocessors and its elements. On the other hand, full system

simulators model features as privileged modes and imitate the function of peripheral components in order to provide support to the operating system. This makes it possible for them to run complicated multithreaded workloads. Simics [11], gem5 [12], RSIM [13] and SimOS [14] use this method. Application- based or user level simulators emphasizes on the user part of applications. Their development and use are much easier as they do not require device-timing models, large disk images and booting of the operating system. However, they are only useful in supporting primary workloads. Graphite [4], Sniper [15], ZSim [16], CMP$im [17] and McSimA+ [18] use this method.

The other classification is based on ***input*** or ***workload***, which include trace-driven and execution-driven simulators. Trace-driven simulators make use of sessions, which were previously recorded when executing simulated applications. It involves recording of operations of memory and other important instructions when running an application in a trace generation environment [19]. This type of environment can be the real hardware, which is the target. However, it can also result from software generations in case the hardware is under development or non-existent. It is necessary to note that irrespective of simplification of execution, trace-driven simulation lacks a vibrant behavior present in multi-threaded programs running on several processors simultaneously. This inconsistency is evident when running this type of simulation on another Symmetric Multi-Processor (SMP) system other than the one used in the trace [19, 20]. Another limitation of trace-driven execution is the time limitation and data storage necessary to record the session(s) [19]. Goldschmidt and Hennessy [19], in their report state that trace-driven simulation should not be used in the simulation of systems depending on timing or parallel system.

Execution-driven simulators implement applications basing on a simulated processor. This type of simulation does not require traces and can be conducted on one machine [19]. This makes the issues of instructions timing and concurrency to be avoided. Accuracy of the simulator is the only affection of correctness of the result from an execution-driven [19]. This method is highly valuable when optimizing a target as it avails all the data, which is in use or being produced by the target [10].

Finally, *detail* is also major category used in classifying simulators. It includes functional (what is done) and timing (when is it done) simulators. Functioning simulators focus on the achievement of the same function as those of the modeled components. Timing simulators, however, attempt to focus on the accurate reproduction of performance or timing characteristics of the respective targets.

4 CSA simulators quality attributes and evaluation parameters

There are different perspectives of balancing various factors in an optimal manner to enhance the functions of the simulator in terms of accuracy, flexibility, performance, degree of details and functionality. The availability of many simulators with specific designs for limited tasks based on definite aspects comes because maximization all of these aspects can be complicated. However, some simulators have a more general approach.

Performance is a term used in measuring the speed of the simulator in completing a specific workload. Its measurement is in terms of slowdowns, which is also known as the host number of instructions is executed for every simulated instruction in the target.

The performance of processors is measured in MIPS (Millions of Instructions Per second) which is using in benchmarking simulators.

The other criterion is **functionality**, which is what the simulator can perform or be able to perform after the modification. A functional simulator can execute a good part of the software that operates on the target, and it will function properly without issues. Functionality also includes which ISA architectures the simulator support? How many and what type of workloads it can run? Can it run multithreaded applications on homogenous and heterogeneous multicore systems?

Accuracy defines how well a simulator repeats the target behavior. And it represents to which degree the simulated target performance is matching with the real system performance. Therefore, it is necessary to note that a more accurate simulator incurs more slowdowns; hence it will simulate a high level of details. Absolute accuracy shows how close the simulation is to the real world whereas relative accuracy shows how correct a model is between diverse configuration settings.

Finally, **flexibility** is also used as a criterion of the simulator as it gives a description of whether the design of the simulator is versatile, it ability to operate from dissimilar hosts and ease of porting it to other hosts among others. It also evaluates the ability of a simulator to have extendable modular design. These entire requirements validate the flexibility of a simulator.

Numerous processor and system simulators are already available as shown in Table I. All listed simulators have their own merits, Howver, a full-system simulator might be particularly useful when the simulation involves heavy I/O activities or extensive OS kernel function support. However, these simulators are relatively slow and make it difficult to isolate the impact of architectural changes from the interaction between hardware and software stacks. Moreover, because they rely on existing OSes, they usually do not support manycore simulations well.

A pure application-level simulation is insufficient, even if I/O activity and time/space sharing are not the main areas of focus. For example, thread scheduling in a manycore processor is important for both performance accuracy and research interests. Thus, it is desirable for application-level simulators to manage threads independently from the host OS and the real hardware on which the simulators run.

For example, Graphite [4] uses less detailed models, such as the one-IPC model, to achieve better simulation speed. Sniper [15] uses better abstraction methods such as interval-based simulation to gain more accuracy with less performance overhead. While these simulators are good for early stage design space explorations, they are not sufficiently accurate for detailed microarchitecture-level studies of manycore architectures. Graphite [4] and Sniper [15] are considered faster simulators because they use parallel simulation to improve the simulation speed. Trace-driven simulations can also be used to trade simulation accuracy for speed. However, these are not suitable for multithreaded applications because the real-time synchronization information is usually lost when using traces. Thus, execution-driven simulations (i.e., simulation through actual application execution) are preferred. On the other hand, full-system simulators model both microarchitecture-level details and OSes. Thus, they sacrifice simulation speed for accuracy. Instead, it is desirable to have a simulator to model manycore microarchitecture details while remaining faster than full-system simulators, which have both hardware and software overhead. Since, simulators targets simulation of thousand-core chips, they have to be a user-level simulator for now. No current mainstream OS scales to thousands of cores, and ISAs also limit the number of cores [16].

.

Table 1. Simulators Categorized by Features. Abbreviations: (FS/A)-Full-System (FS) vs. Application-Level (A), (Free-AC) free Academic Institution, (UIUC/NCSA) Univ. of Illinios/NCSA OS License, (BSD) Berkeley-Style Open Source License, (SS)-Simulation Speed.

	Year, Availability	FS/A	LoD	Simulated Platform	SS	Extensibility	Special Features
Simics	1994,commercial/free AC	FS/A	OoO	UltraSparc,PowerPC, MIPS,ARM	+	API provided	Simulate "everything", heterogeneous nets
gem5	2003, free BSD	FS/A	OoO	Alpha, ARM, SPARC, MIPS, POWER and x86	+	Modular /OO design, Python config. files	Written from scratch, pervasive OO, configurability
RSIM	1997, UIUC/NCSA	FS	OoO	SPARC V9/Solaris	+	Modular design	Simulate adv. CPUs in network
SimOS	1994, freely available	FS	Variable	MIPS/SGI IRIX,Alpha/UNIX	+	Uncertain	Simulate entire system
Graphite	2010, free	A	No	x86	+++	Python config files	Parallel simulator
Sniper	2011, free	A	OoO*	x86	+++	Python config files	Interval simulation
CMP$im	2008, free	A	No	x86	++	Python config files	Memory system simulator
ZSim	2013, free	A	OoO*	x86	+++	Python config	Fast interpreter design
McSimA+	2013, free	A	OoO*	x86	+++	Python config files	Middle ground between FS/A

For example, x86's xAPIC only supports up to 256 cores. The simulators have varying characteristics, strengths and weaknesses, and relevance to different research areas. While most development projects are only a few years old, a simulator such as SimOS can be considered to be too old to be used actively in research now. CMPSim and ZSim have the problem that they are not available publicly, and would be hard to obtain. Some simulators can be integrated with useful timing simulators as – TFsim and GEMS – use a timing-first approach to simulation, where a less extensive model simulates the timing of the processor, which is then later verified by a larger and more complex execution-drive simulator, such as Simics. All the simulators can simulate Chip Multi Processors (CMP) architectures, but only a few can simulate SMT.Industry simulators typically run at a speed of 1 to 10 kHz. Academic simulators, such as gem5 [12], GEMS [7] and PTL-Sim [21] are not truly cycle-accurate compared to real hardware, and therefore they are typically faster, with simulation speeds in the tens to hundreds of Kilo Simulated Instructions Per Second (KIPS). Cycle-accurate simulators face a number of challenges in the multi-core era. First, these simulators are usually single-threaded, and their performance does not increase with increasing core counts. Second, given its slow speed, simulating with large caches becomes increasingly challenging because a slow simulation speed does not allow for simulating huge dynamic instruction counts.

5 Case Study: Sniper

Sniper is a high-speed and accurate x86, parallel simulator. Sniper integrates interval simulation method and expands the general functionality of Graphite simulation

infrastructure; permitting for fast and accurate simulation. Trading off simulation speed for accuracy allows Sniper to have a variety of flexible simulation options when exploring various homogeneous and heterogeneous multi-core architectures.

This study describes the architecture of Sniper Simulator, the key components, configuration of simulation, result tools and different visualization alternatives. To show how to use the sniper simulator practically, this study demonstrates how to start various test runs, outcomes and interpretation of recorded results. The tests of simulation depend on pre-tailored test binaries accompanying the simulator. The sniper simulator is well developed and has a manual for setup configurations. The instructions contained in the manual makes it easy for users to know where to start and how to use the device during operations. The sniper has documentation of reasonable quality in comparison to other similar solutions. When using the sniper simulator a person can do timing simulations for multi-program workloads and multi-threaded, shared memory applications with tens to hundred cores. The speed of performing this operation is higher than the speed of existing simulators. The central feature of the sniper simulator is the core model based on interval simulation, a fast mechanistic core model. Simulating at intervals allows a person to raise the level of abstraction in architectural simulation. This guarantees faster development of the simulator and evaluation times. For example, this happens it "jumps" between miss events called intervals. The sniper simulator is validated against Nehalem and multi-socket Intel Core2 systems thus there are average performance estimation errors within 25 percent at the simulation speed of up to many MIPS [15].

The Sniper simulator and interval core model is vital in the uncore and system level studies requiring many more details than the traditional one-IPC models and when the cycle-accurate simulators are too slow to enable reasonable simulation workloads [22]. Another benefit is that the interval core model supports the creation of Cycles Per Instruction (CPI) stacks. These stacks portray the number of cycles that get lost due to the varying characteristics of the system (i.e., the branch predictor or cache hierarchy). This enables to use the Sniper for characterization of applications and software/ hardware co-design.

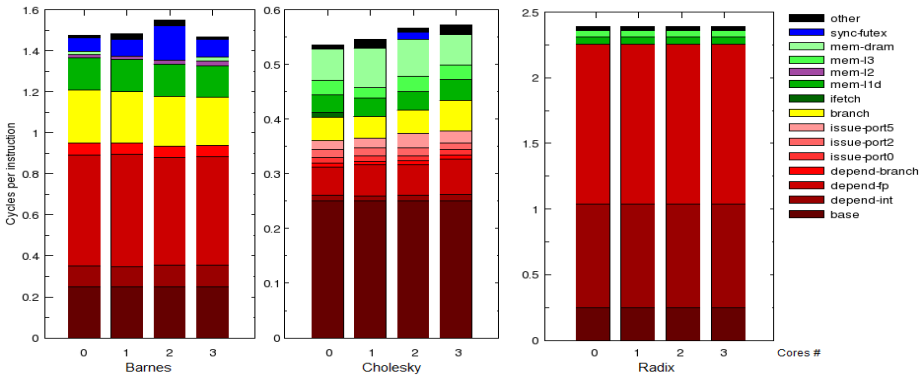

Fig. 1. Cycles Per Instruction stack

5.1 Sniper Configuration

The configuration of the sniper simulator is achievable with command line parameters and configuration files. In practice, the default configuration exists at sniper/config/base.cfg. The script of run-sniper accepts the command line parameters and the evaluation of parameters or configuration files occurs from the left to the right with respect to the command line. Also, newer values override the older values.

5.2 Simulation Results Tools

The sniper simulator creates files at the end of a simulation process; the sim.cfg which has configuration alternatives that are in use. The sim.out file shows the primary statistics and the sim.stats[.sqlite3] has the following set of tool to capture key points of the simulation.**:**

- ***CPI Stacks.*** Interval simulation is unique because allows for the creation of CPI stacks that summarize the place where time is spent. The CPI stack can be described as a stacked bar that shows different components that contribute to the overall performance as shown in fig.1. The base CPI exists at the bottom and shows the meaningful work that is done. The CPI stack is significant when gaining insight in the performance of the application.
- ***Power and Area Stacks*****.** To estimate the power consumption of a program, the sniper simulator integrates with the McPAT power and modeling framework. When the files, sniper/tools/mcpat.py and area.py, are run in the directory containing the sniper output files, power and area stacks can be generated. With respect to this data, the files, area.png and power.png are generated together with the output texts that depict power consumption of the application, broken down by the component. There are options of choosing plot dynamic, static or total power of the chip is for every component (Fig.2).
- ***Visualization.*** The options of Sniper visualizations help users comprehend the behavior of the software that works in the simulator. Since no single visualization can satisfy needs of various users, the simulator has to produce a number of visualizations. Currently, three main groups of visualizations (Fig.3). For example, they include cycle stacks plotted against time, McPAT plotted against time and 3D Time-Cores-IPC visualizations. When a user passes the --viz option to the run-sniper command, the visualizations are generated. The Sniper also creates an extra section of system topology on the produced --viz web page.
- ***Multiple Multi-threaded workloads*****.** One can run multiple, multi threaded benchmarks simultaneously by installing the integrated benchmarks suite first. Thereafter, the user can utilize the benchmarking parameters to show the benchmarks and configurations that should be run. There is no maximum number of threads that a user can run when utilizing the --benchmark parameters. However,

it is worth noting that this mode does not support the ROI handling or the multi-threaded workloads [22].

- ***Scripting.*** The Sniper has a Python interpreter that is used to request for statistics and update the configuration of hardware parameters when the simulator is in use. Consequently, this enables generation of IPC traces and DVFS updates**.**

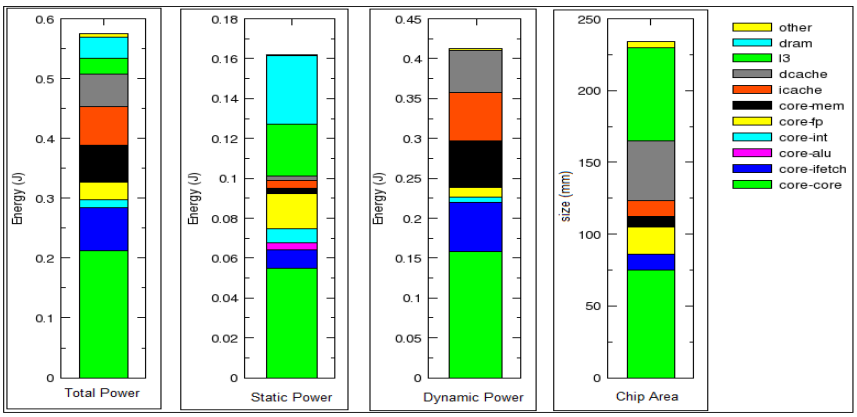

Fig. 2. Power and area stacks

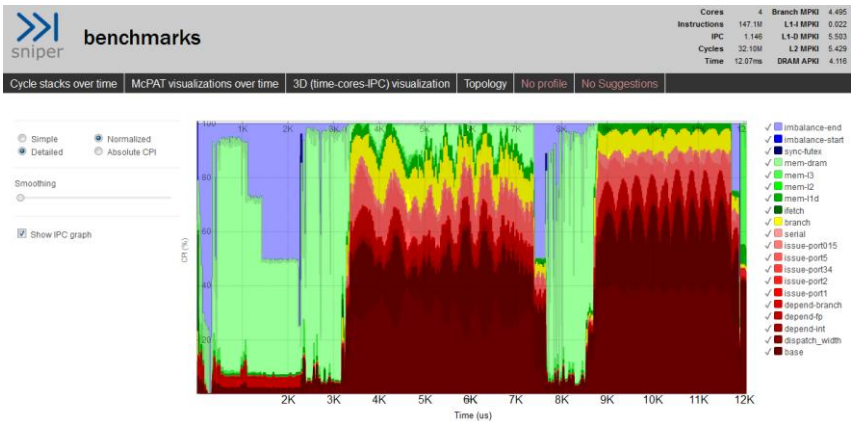

Fig. 3. Cycle stacks plotted over time

5.3 Comparison between Sniper and Graphite

In the interest of comprehending the distinguishing features between graphite and sniper, it is imperative to underline that graphite is the base of sniper whereby it adds, removes and reworks on its features. This work delves on the strengths and weaknesses of different approaches. By design, the default simulator of the Sniper is configured in a similar manner as Nehalem machine the Gainestown micro-architecture. Sniper produces the most significant instruction latencies such as FP and math ops. In the

past, Sniper used the Graphite set up as the starting point but Sniper designers designed today's models.

There are several distinct features between the simulators. To differentiate between Sniper and graphite, graphite has been added the MESI coherence protocol while Sniper uses a shared-version of MSI. Recently, Graphite has also been integrated with ATAC [23] and DSENT [24]. Further on, Graphite supports full-mode that is not in the recent versions of Sniper. Full-mode tolerates the distribution of single application simulation across many machines. This is very handy in handy if one has a large memory footprint that would be large enough to stimulate on a single machine. However, this can make the simulation reduce in case of barrier synchronization is used instead of relaxed synchronization that graphite uses automatically [25]. It is important that the system calls are rooted so that they are executed on behalf of the operator. Sniper uses light-mode entirely, when it passes the system calls onto the OS.

Sniper has more quality visualization features that allow better understanding individual runs with topology information for the architecture being generated. McPAT energy was also incorporated into the visualization and it relays good results when they are run with Sniper core and other models. Further on, Sniper has Python scripting support and a strong link between the SimAPI, scripting and the simulator. Sniper is also compatible with MPI applications as well as the interchange that takes place between different processes through clear virtual-to physical mappings provided by the OS [26]. Sniper had initially added true-DVFS support that allowed each machine to run its own frequency. Sniper database statistics substructures are also sophisticated to allow automatic generation of graphs and reports from the databases. Other features of the Sniper include loop tracer, fault injection, statistically distributed DRAM latencies and micro-op support [22].

6 Conclusion

It is critical to use a simulator tools that model many-core micro architecture details at a faster rate than the full-system simulators characterized by software and hardware overheads. At present, the main challenge is the simulation of thousand-core chips, however, there is no mainstream operating system that can scale to this level. Moreover, the Instruction Set Architectures (ISAs) limits the number of cores thus making it hard to achieve the desired target. In particular, x86's Advanced Programmable Interrupt Controllers (xAPICs) can support maximum 256 cores. At this level of operation, choosing a simulator has to be based on the user-level simulator type. The Sniper simulator is easy to use, and it can adequately balance performance and accuracy trade-offs. However, not all instructions supported by various systems (i.e. the SSE4 and 64-bit x86) are modeled in Sniper.

The number of processes in a system and cores of each packet continues to grow creating the challenge of simulating the growth in system sizes. A rapid growth of multi-core technology combined with much larger cache sizes, requires long and accurate simulations to test the next generation of system designs.

References

1. E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," in *ACM SIGMETRICS Performance Evaluation Review*, 2003, pp. 318-319.
2. R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Comp. Architecture, 2003. Proc. of 30th Annual International Symposium*, 2003, p.84-95.
3. J. Chen, M. Annavaram, and M. Dubois, "SlackSim: a platform for parallel simulations of CMPs on CMPs," *ACM SIGARCH Computer Architecture News*, vol. 37, pp. 20-29, 2009.
4. J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, *et al.*, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1-12.
5. S. S. Mukherjee *et al.*, "Wisconsin Wind Tunnel II: a fast, portable parallel architecture simulator," *Concurrency, IEEE*, vol. 8, pp. 12-20, 2000.
6. A.-M. M. C. Z., "A Survey of Computer System Architecture Simulators, Case Study: Sniper " in *APCASE 2014*, South Kuta, Indonesia, 2014, pp. 014-015.
7. M. M. Martin, *et al.*, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Comp. Arch. News*, v.33, pp. 92-99, 2005.
8. T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "TurboSMARTS: Accurate microarchitecture simulation sampling in minutes," in *ACM SIGMETRICS Performance Evaluation Review*, 2005, pp. 408-409.
9. P. S. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, *et al.*, "SimICS/sun4m: A virtual workstation," in *Proceedings of Usenix Annual Technical Conference*, 1998, pp. 119-130.
10. T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, pp. 59-67, 2002.
11. P. S. Magnusson, *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, pp. 50-58, 2002.
12. N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, *et al.*, "The gem5 simulator," *ACM SIGARCH Comp. Arch. News*, vol.9, pp.1-7, 2011.
13. V. S. Pai, P. Ranganathan, and S. V. Adve, "RSIM: An execution-driven simulator for ILP-based shared-memory multiprocessors and uniprocessors," *Proceedings of the Third Workshop on Computer Architecture Education*, 1997.
14. M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete computer system simulation: The SimOS approach," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 3, pp. 34-43, 1995.
15. T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, p. 52.

16. D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," 2013.
17. A. Jaleel, R. S. Cohn, C.-K. Luk, B. Jacob, "CMP$im: A Pin-based on-the-fly multi-core cache simulator," *Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), ISCA co-located,* 2008, pp.28-36.
18. J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," *ISPASS,* Apr 2013.
19. S. R. Goldschmidt and J. L. Hennessy, *The accuracy of trace-driven simulations of multiprocessors* vol. 21: ACM, 1993.
20. M. D. Dikaiakos, A. Rogers, and K. Steiglitz, "Fast: A functional algorithm simulation testbed," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems,MASCOTS'94., Proceedings of the Second International Workshop on*, 1994, pp. 142-146.
21. M. T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *Performance Analysis of Systems & Software, ISPASS 2007. IEEE International Symposium on*, 2007, pp. 23-34.
22. T. E. Carlson and W. Heirman, "The Sniper User Manual," Nov 13, 2013. Available: http://groups.csail.mit.edu/carbon/?page_id=62
23. *MIT-DSENT*. Available: https://sites.google.com/site/mitdsent
24. (May 2014). *Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations*. Available: *http://snipersim.org/w/Paper:Sc2011Carlson*
25. (May 2014). *The Sniper Multi-Core Simulator*. Available: http://snipersim.org/w/The_Sniper_Multi-Core_Simulator