embeddedworld2017
Exhibition&Conference
...it's a smarter world

Fast Fault Injection to Evaluate Multicore Systems Soft Error Reliability

Felipe Rosa[1], Luciano Ost[2], Ricardo Reis[1], Simon Davidmann[3], Larry Lapides[3]
[1]UFRGS - Instituto de Informatica - PGMicro/PPGC
[2]Department of Engineering - University of Leicester
[3]Imperas Software Ltd.
{frdarosa, reis}@inf.ufrgs.br, luciano.ost@le.ac.uk, {simond, larryl}@imperas.com

***Abstract*— The increasing complexity of processors allied to the continuous technology shrink is making multicore-based systems more susceptible to soft errors. The high cost and time inherent to hardware-based fault injection approaches make the more efficient simulation-based fault injection frameworks crucial to test reliability. This paper proposes a fast, flexible fault injector framework which supports parallel instruction accurate simulation to boost up the fault injection process. Fault injection campaigns were performed on ARM processors, considering a Linux Kernel and benchmarks with up to 220 million object code instructions. Results have shown the injection of faults at speeds up to 1550 MIPS. This enables users to identify errors and exceptions according to different criteria and classifications.**

Keywords—component; soft error; fault injection simulation; multicore systems.

I. INTRODUCTION

The increasing computing capacity of multicore components like processors and graphics processing units (GPUs) offers new opportunities for embedded and high performance computing (HPC) domains. The progressively growing computing capacity of multicore-based systems enables the efficient performance of complex application workloads at a lower power consumption compared to traditional single core solutions. Such efficiency and the ever-increasing complexity of application workloads encourage industry to integrate more and more computing components into the same system. The number of computing components employed in large-scale HPC systems already exceeds a million cores [1], while 1000-cores on-chip platforms are available in the embedded community [2].

Beyond the massive number of cores, the increasing computing capacity, as well as the number of internal memory cells (e.g. registers, internal memory, etc,) inherent to emerging processor architectures, is making large-scale systems more vulnerable to both hard and soft errors [3], [4]. Moreover, to meet emerging performance and power requirements, the underlying processors usually run in aggressive clock frequencies and multiple voltage domains, increasing their susceptibility to soft errors, such as the ones caused by radiation effects. The occurrence of soft errors or Single Event Effects (SEEs) may cause critical failures on system behavior, which may lead to financial or human life losses as already reported in [5], [6]. While a rate of 280 soft errors per day has been observed during the flight of a spacecraft [7], electronic computing systems working at ground level are expected to experience at least one soft error per day in near future [8]. The growing susceptibility of multicore systems to SEEs necessarily calls for novel cost-effective tools to assess the soft error resilience of underlying multicore components with complex software stacks (operating system-OS, drivers, etc.) early in the design phase.

With this trend in mind, researchers are investigating new fault injector techniques as well as proposing new tools to evaluate the occurrence of SEEs in commercial state of the art processors. In this context, the use of virtual platform frameworks is attractive due to their simulation performance and design flexibility (i.e. support for a large number of component models, compilers, and debugging facilities). Due to the high simulation speed (typically at hundreds of MIPS), virtual platform simulators based on just in time (JIT) dynamic binary translation appear to have an advantage over event-driven simulators. However, this simulation performance comes at the cost of limited microarchitecture exploration support and timing accuracy. The resulting scenario poses a major challenging question: can we rely on soft error analysis produced from JIT-based frameworks?

To address the gap between the available fault injection tools and the industry requirements, this paper describes the development of a fault injector module (FIM) that was assembled with OVPsim [9], [10], which relies on JIT dynamic binary translation technology. Aiming at answering the above challenging question on JIT simulation credibility, the developed FIM was integrated into gem5 [11], which is an event-driven virtual platform framework that targets microarchitecture exploration.

The main contributions of this work are the following:

- Proposal of a fast and flexible fault injector framework, called OVPsim-FIM, which supports the

analysis of complex systems considering more than 100 commercial processor models.

- Soft error analysis consistency of the proposed OVPsim-FIM with respect to gem5 Full System Simulation.
- The extensive OVPsim-FIM evaluation by using several and large scale benchmarks.

The rest of this paper is organized as follows. Section II presents related works in simulation based fault injection frameworks developed on virtual platforms. The concepts and the main features of developed fault injection frameworks are described in Section III. In Section IV the performance efficiency of proposed framework is evaluated under a set of fault injection experiments, considering several benchmarks and processor architectures. Section V describes fault injection campaigns performed in ARM Cortex family processors. Afterwards, conclusions and perspectives are discussed in Section VI.

II. Related Works in Virutal Platform-based Fault Injection Simulators

Early and fast soft error sensitivity evaluation is of utmost importance for better identification of most common source of errors (e.g. a single-bit error in a memory area), which may lead to either a simple data corruption or a serious system failure [2]. In this direction, Authors in [7] present the Relyzer, a hybrid simulation framework for SPARC core using Simics [8] and gem5 [9] simulators coupled with a pruning technique to reduce injected faults. In [10], a fault injection framework based on QEMU is proposed. Faults are injected in an X86 architecture running applications in a Real-Time Operating System (RTEMS). During the experiment, 8,000 faults were injected in 8.7 hours, given an average of less than one fault per second.

More recently, authors in [11] propose the GeFIN tool, a gem5-based fault injection framework. In this work faults were injected, randomly in time, in general-purpose registers, caches control registers, and other components. The experimental setup includes only the execution of 10 bare metal benchmarks selected from the MiBench [12].

Most reviewed approaches consider only small scenarios and a single-core processor or specific ISA [7]. Exploration of soft error reliability of single-core architectures has been successfully supported over the last decades. However, the assessment of multicore architecture soft error resilience strongly requires complementary modelling and simulation mechanisms to manage other aspects such as resource sharing, memory allocation and data dependencies. Further, such works typically report best-case simulation performances of 2-3 MIPS, allowing 33 fault injections per second considering a supercomputer [7].

Different from reviewed works, reported fault injection analysis include more than 1.1 million fault campaigns varying the number of CPU cores of an ARM processor, executing a Linux Kernel and large scale and realistic benchmark applications targeting high performance systems. Moreover, this work is the first that reports and discusses the soft error analysis consistency of a JIT-based VP simulator against an event-driven full system simulator.

III. gem5-FIM and OVPsim-FIM Simulators

The gem5 simulator [11] was selected among the available cycle-accurate simulators due to its open and free availability as well as its support for the ARM Cortex-A architectures with three execution models. Additionally, it is a well-known simulator used in many research projects, which increases its acceptability within the community. The proposed gem5-FIM is responsible for creating and injecting faults, capturing unexpected events and the required information (e.g. instruction count, memory dump) used to generate detailed soft error analysis. For the sake of simplicity, any kind of application behavior divergence is considered as a soft error, which is classified according to [18].

We model SEEs through single bit-flips generated randomly in any available general purpose register during the software stack execution (i.e. OS, drivers and applications). Intentionally, we do not consider faults injected directly into OS routines, nevertheless OS calls may affect the application behavior. The implemented approach analyses the application behavior considering the whole system execution environment, i.e. processor model, optimization flags, compiler, cross-compiler, libraries, and a Linux kernel (3.12.0) built with the same cross-compiler.

As mentioned before, the gem5 is an event-based cycle-accurate simulator, enabling a finer granularity when selecting the fault insertion time. In this regard, the proposed implementation uses the instruction count as a temporal reference. The gem5 simulator employs Python scripts to control the simulation flow and C++ modules to the microarchitectural simulation. Fig. 1 displays the main FIM components.

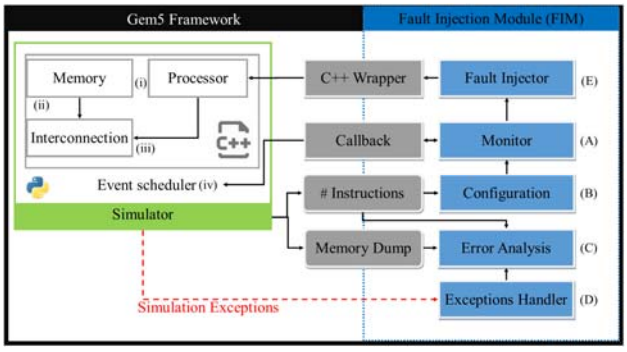


Fig. 1 gem5 fault injection module main components.

Differently from OVPsim-FIM, the gem5 detailed simulator provides sophisticated memory timing and cache coherency protocols. However, the gem5 atomic simulator emulates the memory and cache using a single tick access mechanism, therefore slightly differing from the gem5 detailed simulator regarding the execution time and cache activity. The complex timing associated with cache misses or the wrong branch speculations leads to pipeline flushes. Thus, the speculatively executed or partially executed instructions are discarded, and the pipeline will fetch new instructions.

This event will re-execute instructions and overwrite the current pipeline context including any effect arising from a fault injection in this period.

OVPsim does not provide microarchitectural modeling (e.g. pipeline, decoder, reorder buffer) and so it does not emulate a register renaming hardware. Reads and writes are always guaranteed to be correct in an instruction-accurate simulator (i.e. the previous instruction always completes before the next start), thus removing the data hazards originating from the pipeline access to data before the write-back stage update. The gem5 atomic simulator models some microarchitectural aspects, for instance, not including the reorder buffer or the renaming register module. Thus, inner transactions occur in a single tick, and consequently every pipeline stage completes before the next instruction, similarly to OVPsim.

A. *Fault Injection Simulation Flow*

Fig. 2 illustrates a five-phase fault injection flow, which is supported in both simulators. In the first phase, named *Golden Execution*, is the compilation and execution of the application in an unchanged simulator to verify its correctness and also to extract the essential information (e.g. internal state and memory map).

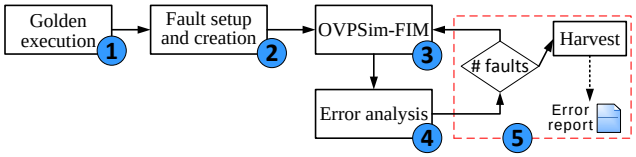


Fig. 2 Simulation phases to accomplish one fault campaign.

The second phase involves fault creation. In Fig. 1 (B), as previously reported, we deploy a random generation scheme (i.e. randomly selecting the insertion time, the location, and the register bit) since it covers most faults at a low computational cost. Our solution selects a random injection time based on the final instruction count extracted in phase 1, the locations (e.g. registers, memory address) are also defined randomly. This solution requires a bit pattern, considering a 32-bit processor such as the adopted Cortex A-9, all bits are set to '0' except the targeted bit. For instance, to change the second least significant bit of a given register requires the 0x00000002 pattern.

The next phase includes the fault injection campaign. At the first step, the fault monitor, Fig. 1 (A), verifies the number of executed instructions until it reaches the fault injection insertion time. Triggering the fault injector, Fig. 1 (E). and at this moment accessing the targeted register. The bit pattern over the current value using an exclusive OR operation (XOR). As an example, suppose that the original value is 0x00000009, and we aim to flip the fourth least significant bit. For this purpose, the module performs a XOR operation between the first 0x00000009 and the pattern 0x00000008, which generates 0x00000001 as expected result from flipping the fourth bit.

After fault insertion, the application behavior may lead to processor or OS exceptions, which may arise in simulation driving the simulation to a halt. To observe this behavior, we deploy a component, Fig. 1 (D), to deal with such unexpected events at run-time. In some cases, the error affects the application control flow without triggering an immediate error, exception, or exiting the application (i.e. the application enters into an infinite loop). To reduce the simulation overhead, in this case, we consider applications as incorrect after executing twice the number of instructions executed in its faultless Golden Execution.

The error analysis (phase 4) comprises comparisons of each application running under fault injection with the golden run to detect arising errors, Fig. 1 (C). For the purpose of this work, we consider an error when a fault leads the application to inconsistent control flow behavior or the data results. Additionally, we deploy the error classification proposed in [18] with five groups:

- **Vanished**, no fault traces are left;
- **Application** Output Not Affected (ONA), the resulting memory is not modified. However, one or more remaining bits of the architectural state are incorrect;
- **Application output mismatch** (OMM), the application terminates without any error indication. However, the resulting memory is affected;
- **Unexpected termination** (UT), the application terminates abnormally with an error indication;
- **Hang**, the application does not finish, requiring a preemptive remove.

The final phase consolidates the set of injected faults information in a single report. Also, in large-scale fault injection campaigns, this phase also handles the synchronization between independent simulations.

IV. Experimental Setup

The experimental setup aims to study the precision of OVPsim-FIM (instruction-accurate) against the gem5-FIM (cycle-accurate). It comprises the evaluation of six applications displayed on TABLE I and selected from among the Rodinia Benchmark suite [19].

TABLE I - Experimental Setup Applications.

#	Name	Domain
A	bfs	Graph Algorithms
B	hotspot	Physics Simulation
C	hotspot3d	Physics Simulation
D	Needleman-Wunsch (NW)	Bioinformatics
E	srad v1	Image Processing
F	streamcluster	Data Mining

The applications are built using an almost identical environment configuration to ensure the fairest comparison as possible. Both simulation flows are generated using the same optimization flags, compiler, cross-compiler, libraries, and a Linux kernel (3.12.0) build with the same cross-compiler. As aforementioned the gem5 has three execution models,

however, we deploy only the most and least precise models, respectively the detailed and atomic models. Thus, a given application is executed in three FIM modes: One using OVPsim and the two gem5 – atomic and detailed – models. The slowest simulator mode (i.e. the gem5 detailed model) bounds the fault injection campaign length even when employing a high-performance computer (HPC), since the required simulation hours for a larger fault injection campaign is infeasible. For instance, the application hotspot3D using the smallest input matrix available (i.e. 64x64x64) executes approximately 220 million instructions to until complete. Their simulation time (using the latest Intel i7) for the OVPsim, gem5 atomic and gem5 detailed respectively are: couple seconds, 136 seconds, and 1438 seconds. When regarding an 8000 fault injection campaign, the most optimistic total simulation time estimates is 3600 hours, where the OVPsim-FIM accounts for only 2.2 hours, the gem5-FIM atomic for 302 hours, and the 3296 remaining hours just for the gem5-FIM detailed. Observe that we do not consider the Golden Execution phase, eventual synchronizations, job scheduling, storage access or speed degradation due to other processes in simultaneous multithreading processors. Additionally, the simulation time does not take in account the Linux boot, which is always restored from a checkpoint. This work exceeds 100 thousand simulation hours in the HPC cluster ALICE from the University of Leicester which counts with more than 5,000 processors.

V. Results

The application fault injection campaign extends to 8,000 fault injections for the three FIM modes using the flow described in the previous section. Thus, it executes 24,000 times randomly assigned faults running on independent platforms. Fig. 3 displays the six-application error analysis using the five classifications proposed by Cho et al. in the three FIM modes OVPsim, gem5 Atomic, and gem5 Detailed respectively.

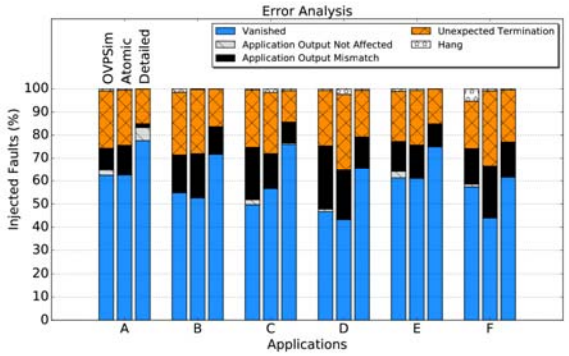


Fig. 3 Error Analysis for the six application over the three FIM modes. The application letter is referred in TABLE I.

The gem5 Detailed presents a distinct behavior for all investigated applications, revealing a noticeable higher fault masking rate than the OVPsim and the gem5 Atomic. The register file internal organization could cause this masking effect as it tries to mimic the register renaming hardware technique when adopting the most accurate model. The mapping will overwrite some injected bit-flips in the registers before the next read, thus suppressing any possible fault effect in the software. The streamcluster application has the smaller deviation in comparison with the detailed model. The streamcluster algorithm originates from the data mining domain, and a significant percentage of its operations consists of reads [20]. Consequently, reducing the possible of read-after-write hazards in the register file, thus approximating both OVPSIM-FIM and gem5-FIM detailed.

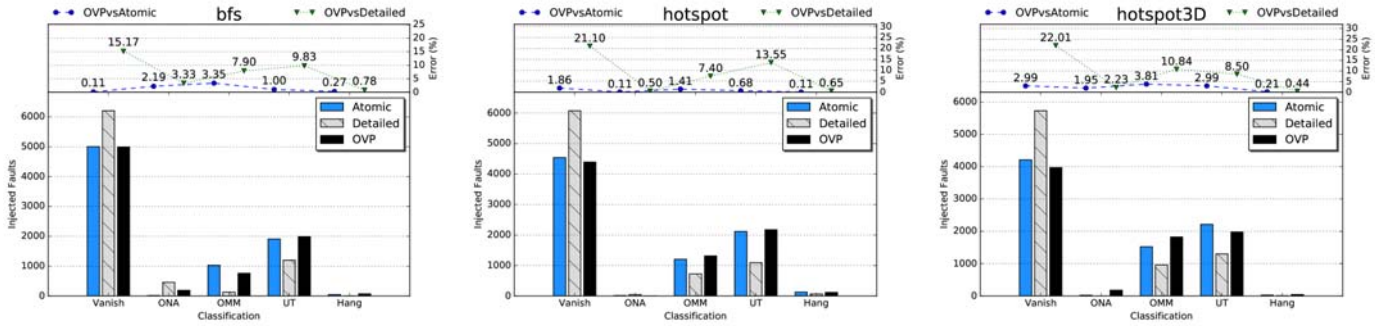
Fig. 4 isolates each error classification in the individual application and displays the two relative errors on top: OVPsim versus gem5 Atomic (dotted blue) and OVPsim versus gem5 Detailed (dashed green). In the first three benchmarks (i.e. A, B, and C) the OVPSIM-FIM response is more closely related to the gem5 Atomic. In contrast, the latter three applications achieve a greater proximity with the gem5 Detailed when considering the applications and the exit statement (i.e. without being externally finished).

VI. Conclusion

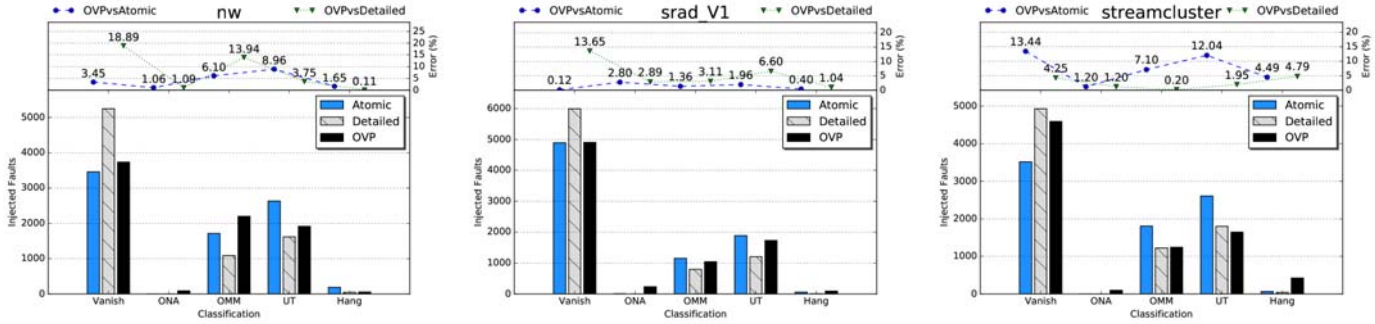
This paper evaluates the consistency of using a JIT-based fault injection simulator to perform soft error reliability analysis when compared to an event-driven full system simulator. Although the percentage of errors due to fault injection is in some cases higher than 20% when compared to the reference gem5-FIM, the authors conclude that these errors are acceptable and are not a hindrance to evaluating soft error at early design phases. Further, given the remarkable achieved speedup, the utilization of JIT-based FIM appears promising since it can also be used for comparison among different processor models, ISAs and benchmarks.

Acknowledgment

This research used the ALICE High Performance Computing Facility at the University of Leicester. The authors would like to thank Imperas Software Ltd. and Open Virtual Platforms for their support and access to their models and simulator.



(A) BFS (B) hotspot (C) hotspot 3D



(D) nw (E) srad_v1 (F) streamcluster

Fig. 4 Individual error classification for the six benchmark applications. The application letter is referred in TABLE I.

References

[1] "TOP500 Supercomputer." [Online]. Available: https://www.top500.org/. [Accessed: 05-Mar-2017].

[2] G. Abich, M. G. Mandelli, F. R. Rosa, F. Moraes, L. Ost, and R. Reis, "Extending FreeRTOS to support dynamic and distributed mapping in multiprocessor systems," in *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2016, pp. 712–715.

[3] M. Snir *et al.*, "Addressing Failures in Exascale Computing," *Int J High Perform Comput Appl*, vol. 28, no. 2, pp. 129–173, May 2014.

[4] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.

[5] "Toyota Case: Single Bit Flip That Killed | EE Times," *EETimes*. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1319903. [Accessed: 18-Feb-2015].

[6] E. Normand, "Single event upset at ground level," *IEEE Trans. Nucl. Sci.*, vol. 43, no. 6, pp. 2742–2750, Dec. 1996.

[7] G. M. Swift and S. M. Guertin, "In-flight observations of multiple-bit upset in DRAMs," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2386–2391, Dec. 2000.

[8] T. Granlund, B. Granbom, and N. Olsson, "Soft error rate increase for new generations of SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 6, pp. 2065–2068, Dec. 2003.

[9] F. Rosa, F. Kastensmidt, R. Reis, and L. Ost, "A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability," in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2015, pp. 211–214.

[10] "Open Virtual Platforms (OVP)." [Online]. Available: http://www.ovpworld.org/.

[11] N. Binkert *et al.*, "The Gem5 Simulator," *SIGARCH Comput Arch. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[12] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting Application-level Fault Equivalence to Analyze Application Resiliency to Transient Faults," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2012, pp. 123–134.

[13] P. S. Magnusson *et al.*, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[14] F. de Aguiar Geissler, F. Lima Kastensmidt, and J. E. Pereira Souza, "Soft error injection methodology based on QEMU software platform," in *Test Workshop - LATW, 2014 15th Latin American*, 2014, pp. 1–5.

[15] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas, "GemFI: A Fault Injection Tool for Studying the Behavior of Applications on Unreliable Substrates," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 622–629.

[16] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith, "Unprotected Computing: A Large-Scale Study of DRAM Raw Error Rate on a Supercomputer," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, Los Alamitos, CA, USA, pp. 645–655.

[17] 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 *2001 IEEE International Workshop on Workload Characterization, 2001. WWC-4*, 2001, pp. 3–14.

[18] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *2013 50th ACM / EDAC / IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10.

[19] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2008, pp. 72–81.