Precise Evaluation of the Fault Sensitivity of OOO Superscalar Processors

Rafael Billig Tonetto, Gabriel L. Nazar, Antonio Carlos Schneider Beck

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
{rafael.tonetto, glnazar, caco}@inf.ufrgs.br

***Abstract*—Since superscalar processors lead the market, their resiliency evaluation by means of fault injection grows in importance. Fault injection strategies usually trade-off their levels of accuracy: low-level HW-based methods are accurate, but very expensive, need special equipment and the actual hardware, and lack controllability; while high-level simulation-based strategies are flexible, fast, easily accessible and have high controllability, but are not accurate since they are based on models that do not always reflect the low-level implementation, mainly when it comes to complex designs like out-of-order multiple-issue processors. In this work, we propose a cycle-accurate fault injection platform for superscalar processors, which has a smart checkpointing mechanism to accelerate injection time, attenuating the shortcomings imposed by the aforementioned fault injection methods while providing the same level of abstraction as detailed RTL models. Leveraging from this new platform, we evaluate a complex and parameterizable Out-of-Order processor (BOOM) by experimenting with different issue widths and analyzing the sensitivity of several hardware structures of the processor.**

***Index Terms*—Fault injection, superscalar processor, register-transfer level**

I. Introduction

As the continuous technology scaling is entailing an ever-increasing rate of soft errors caused by transient faults [1], reliability evaluation becomes of crucial importance in the early phases of design.

Several techniques for error-resilience evaluation can be used for this purpose, such as the radiation-based/accelerated testing, and fault injection (FI), which can be roughly classified in HW-based, software-implemented fault injection (SWIFI), emulation-based, and techniques based on high-level simulators. Each of these methods can be traded off in terms of injection time, cost, controllability, availability, and accuracy [2].

In general, the "closer to the hardware", the more accurate the resilience evaluation will be, since low-level techniques intrinsically mean a more detailed representation of the hardware structures. In terms of accuracy, the accelerated testing and HW-based strategies perform better, since the real hardware is used in the process. However, both techniques may be impractical since they are expensive, not controllable, and the physical implementation is demanded beforehand.

Another approaches are the SWIFI and simulation-based FI, which are simple, cheap, and controllable. However, the SWIFI method usually works by executing special software code to inject faults in registers, memory or application code [3] at instruction-level, which is a coarse-grained method that cannot accurately estimate the levels of sensitivity of the different hardware structures of a processor. Similarly, simulation-based strategies commonly adopt high-level/functional simulators that work at instruction-level and only program-visible structures can be fault injected, this strategy is coarse-grained.

While this discussion applies to any hardware structure or processor, superscalar designs are a particular and very representative case because they dominate the market as they deliver the best performance in terms of instructions per cycle (IPC) at a reasonable cost. Therefore, fault tolerance evaluation for such designs is becoming more and more important. However, since low-level hardware descriptions of fully functional and complex versions of superscalar processors were not available until a short time ago, designers had to choose between HW-based/accelerated testing (restricted, expensive, with low controllability) and SWIFI or high-level/functional simulators that are fast and flexible, but inaccurate. Moreover, the different models of operation and implementation of the several hardware structures that compose a complex superscalar processor make the resiliency of each structure to be considerably different from each other. Therefore, when implementing fault tolerance mechanisms, one should have deep knowledge concerning these different levels of resiliency of particular structures, so that the most sensitive parts of the processor can be properly protected with minimal overhead.

Considering all of these aspects, we propose a framework to evaluate the sensitivity of the Superscalar Berkeley Out-of-Order Machine (BOOM) processor. The framework is a C++ simulation-based platform, which uses a high-level language to describe the hardware at RTL. With that, it is flexible, with a high level of controllability, and fast enough so a representative number of faults can be injected and, most importantly, still providing high accuracy by modeling the processor at low-level. Such a tool provides the benefits from both the hardware (representative) and the simulator (controllable, low cost, accessible) domains. Leveraging from this tool, and as an example usage of the platform, we perform FI into several parts of three different configurations of BOOM at low-level and show the impact of soft errors on particular structures of the processor when different workloads execute.

This paper is organized as follows: Section II makes a short review on previous works on FI strategies. Section III describes the proposed platform. Finally, sections IV and V

978-3-9819263-0-9/DATE18/©2018 EDAA

presents the methodology and evaluation of the framework.

II. Related Work

Several resiliency evaluation strategies have been developed over the years. Authors in [4] introduced the concept of Architectural Vulnerability Factor (AVF), which is defined as the probability that a fault in a particular structure will result in a failure. AVF is estimated by tracking the fraction of bits in a processor that is *not* necessary for the correct execution of the workload. The bits that are necessary for the correct execution of the program are termed Architecturally Correct Execution (ACE) bits, as opposed to the un-ACE bits, which are bits that cannot affect the correctness of the program regardless whether faults affect them or not.

ACE analysis has advantages over FI in that the application has to be executed only once without requiring the physical implementation. Several resiliency evaluation tools based on this technique can be found particularly for superscalar processors (e.g., [4] [5] [6] [7]). However, ACE analysis is a very conservative technique because it is based on tracking the fraction of un-ACE bits by means of abstract performance models where all of the bits that cannot be identified as un-ACE are conservatively assumed as ACE, hence this technique yields a lower bound estimate for the resiliency of the system. In fact, authors in [7] report a lower bound in resiliency of up to 3.5× when comparing ACE analysis to a detailed FI campaign.

As an alternative, some adopt the FI strategy in low-level models of the system. In [8], authors perform FI in a high-performance processor by using its RTL Verilog model. The detailed RTL description of the system yields remarkable accuracy, however, there is a performance penalty in terms of injection time.

High-level/functional simulators are also commonly used as a means of resiliency evaluation (e.g., [3] [9] [10]). Advantages are these tools are fast due to their high levels of representativeness, however, because they do not model the system at RTL, there is an equivalent loss in detail.

In [11], on the other hand, authors present a microarchitecture-accurate fault injector built upon the gem5 simulator (GeFIN). Even though gem5 exposes a significant part of the microarchitecture by modeling array-based/storage components, it does not model the system at RTL, so not providing access to intermediate flip-flops present in the hardware that compose the control blocks of microprocessors, for instance. In [12], the accuracy of fault resiliency evaluations with gem5 is quantified by comparing the gem5 assessment to a detailed FI assessment at RTL. Even though gem5 is faster (about two orders of magnitude) than the RTL model, results show a difference in the levels of resiliency of up to 10% for the register file, and 20% for the L1D, for example.

As a workaround, we propose a flexible and RTL-accurate platform that models a superscalar processor in a C++ simulator, which has a checkpointing mechanism that attenuates the performance penalty imposed by the RTL model. Differently from previous works, we inject faults in a parameterizable superscalar processor at RTL with high controllability and acceptable performance, so we can analyze the impact of FIs in structures of the processor at low-level. The next section details the infrastructure of the platform.

III. Framework Overview

BOOM is a parameterizable OoO superscalar processor [13] written in Chisel (Constructing Hardware in Scala Embedded Language), a hardware construction language derived from the Scala programming language [14]. The Chisel version used in our framework works by translating its Scala source code into both Verilog and C++ equivalent source codes.

The C++ Chisel-exported code defines an RTL and cycle-accurate simulator, so the exported Verilog and C++ models are fully equivalent in terms of representativeness. This feature shortens the gap between the HW-based and high-level/simulation-based traditional fault injection tools by providing access to the processor at RTL by means of a cycle-accurate simulator with complete control over where and when faults are injected.

Because the fault injector works at RTL, there is a performance penalty in terms of injection time. Therefore, we developed a checkpointing mechanism that allows it to skip parts of the execution that are unimportant for the fault resiliency evaluation. It works by saving and restoring the state of the registers, caches, and memory considering a chosen interval between them.

As opposed to GeFIN, in [15], our checkpointing strategy saves/restores the entire state of the processor, which includes both the data and control path registers, not only array-based structures. Therefore, there is no single loss of micro-architectural state when the context of the processors is restored to a given cycle, which implies that our checkpointing mechanism will not affect the accuracy of the fault resiliency evaluation by any means.

A. Fault Injection Life Cycle

The complete FI process is depicted in Figure 1. Collecting golden (fault-free) checkpoints and profiling the application must be the first step in the FI campaign. Application profiling works by collecting the contents of the main memory when the application finishes and the number of cycles taken by the application to execute in a *fault-free* run. Once all the necessary golden checkpoints are collected, the *faulty* mode starts, when the actual faults are injected in the processor. The *faulty* mode is the framework's bottleneck in terms of execution time since the simulator is supposed to run for several times in this mode, as thousands of faults have to be injected. Therefore, performance becomes of crucial importance, and the checkpointing mechanism is used to speed up the injection time.

As shown in Figures 1 and 2, the checkpointing mechanism works in two distinct ways: first, it fast-forwards the application and, afterwards, it early-stops the execution in case the fault is masked. As far as we know, this speedup trick was

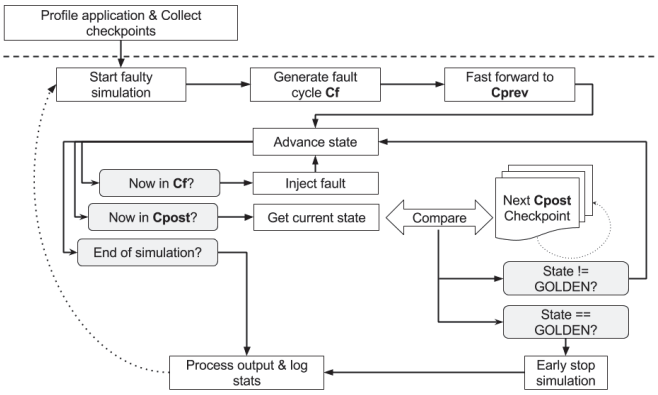

Figure 1: The complete life cycle of a fault injection campaign.

first coined in [15], and we adopted a similar strategy due to its remarkable benefits.

In short, the checkpointing mechanism works as follow:

1) **Fast-forwarding**: Since the processor's state before the fault is injected is of no importance, it can be skipped without any loss of information. The application is fast-forwarded to the nearest checkpoint available right before the FI cycle (the *Cprev* checkpoint).
2) **Early-stopping**: After the fault is injected, it is possible that the faulty bit is overwritten before it is read, preventing the fault from propagating to other hardware structures. This masking effect can be detected in any post-fault checkpoint (*Cpost*) by comparing the state of the processor to its corresponding golden state. In case the fault is masked, the application can be halted since no failure can arise. For empirical reasons, early-stopping is handled by comparing a maximum of five *Cposts*, which are the next four available checkpoints right after the fault cycle and, lastly, the checkpoint located right in half of the remaining application execution, as depicted in Figure 2.

Once the application ends execution, the simulation results are processed by comparing the final contents of the memory to the golden one. For each fault trial, four outcomes are possible: Masked fault; Silent Data Corruption (SDC); timeout; or, simulation crash.

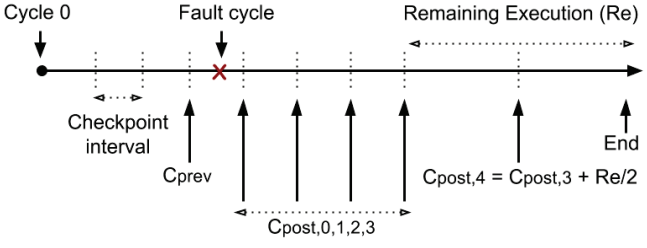

Figure 2: The checkpointing mechanism functionality.

B. Checkpointing Speedup

We evaluated our checkpointing mechanism by choosing a checkpoint interval of 500 cycles. For this case, it is worth mentioning that the faults masked in about 70% of the injection trials before 1k cycles, which highly justifies the benefits of early-stopping the execution. Finally, enabling both fast-forwarding and early-stopping during injection campaigns led to a speedup up to 4.3×.

IV. Experimental Methodology

As an extensive example usage of our parameterizable platform, we performed FIs in particular structures of three different configurations of BOOM (single-, dual-, and quad-issue parameterizations). We injected single event upsets (SEU) in several hardware structures: the register file (RF), the registers used by the register renaming logic (RENAME), the reorder buffer (ROB), the issue unit (IU), the branch reorder buffer (BROB), which is a reorder buffer responsible for updating the prediction structures in-order; and other registers used during the execution stage (EXE), which includes the flip-flops in the ALUs, the FPU, the load/store queue and the bypass network logic. Because our platform works at RTL, it is worth noting that all of the sequential logic encompassing the processors is potentially subject to faults, which includes the intermediate flip-flops of the sequential circuits of the hardware structures, for example.

Fault injections were performed in each structure until their sensitivities stabilized. We term the *sensitivity* of a structure the fraction of failures (SDC's, crashes, or timeouts) raised due to faults injected in the structure.

V. Platform Validation

As a means of validation of our platform, Figures 3 and 4 show the preliminary sensitivities obtained for the IU, EXE, RF, and RENAME structures for each of the running applications. Applications in Figure 3 are sorted in ascending order by IPC. Note how the trends in the sensitivities roughly accompany the IPC, as bigger IPCs for these structures implies in bigger utilization, which may reflect in a bigger fraction of ACE bits.

BOOM makes use of the register renaming strategy. At microarchitectural-level, the RENAME structure maps logical

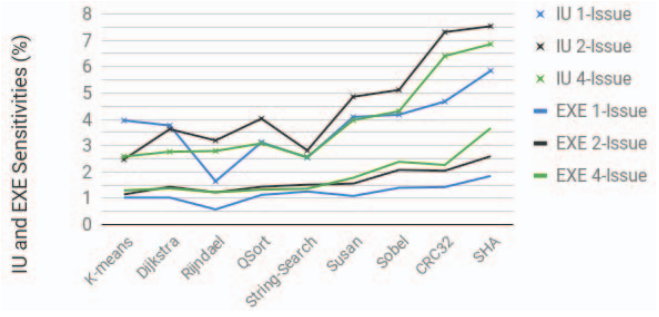

Figure 3: IU and EXE sensitivities for the three cores.

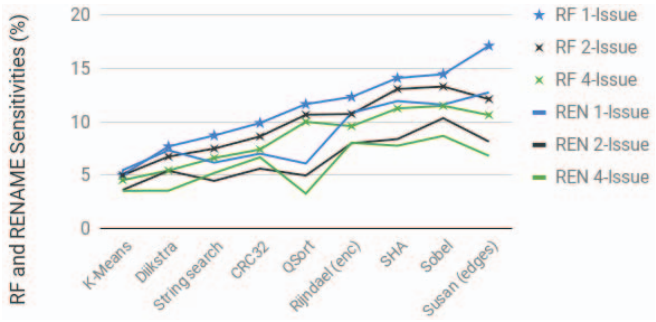

Figure 4: RF and RENAME sensitivities for the three cores.

registers to physical ones in RF, which in turn updates the RENAME state during instruction commit. Therefore, faults injected in RENAME tend to propagate to RF, and vice-versa. For this reason, the characterization to faults for both structures tends to be similar. The benchmarks in Figure 4 are sorted according to the average utilization of the RF, in ascending order, which is roughly accompanied by the increase in its sensitivity. We estimated the average utilization of the RF by monitoring some hardware structures used by BOOM in the register renaming process.

Figure 5 shows the global sensitivity, averaged for all the benchmarks, for the three cores. Recall that our platform performs fault injections in both data and control paths of the processor. Because we do not only perform fault injections in storage structures, the fraction of timeouts closely matches the fraction of SDCs due to injections in the control path, thus justifying the importance of resiliency evaluation by means of low-level platforms.

Because the proposed platform allows it to experiment with parameterizable processors, we can easily experiment with a vast design space of different complex cores, for which the results show, have significant differences in their levels of resiliency for particular structures. Moreover, the high overheads imposed by fault tolerant mechanisms foster the need for evaluations of complex systems at low-level.

VI. Conclusion and Future Work

We implemented an RTL platform for the resiliency evaluation of the BOOM processor. The RTL accuracy of the tool shortens the gap between HW-implemented methods and high-level simulated fault injection. A checkpointing mechanism reduces the performance penalty imposed by the RTL model of the processor. We evaluated three different configurations of BOOM (single-, dual-, and quad-issue) as a means of validation of the platform. Future work will use the platform to evaluate the efficacy of software-based fault tolerance mechanisms.

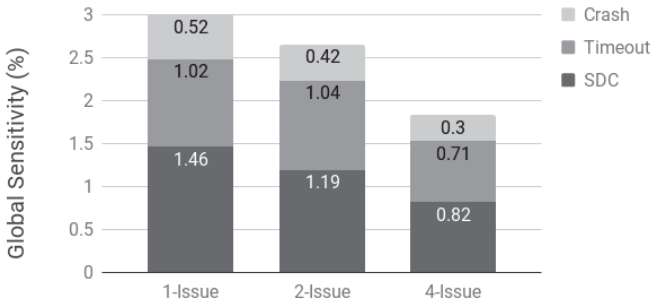

Figure 5: Global sensitivity for the three cores.

VII. Acknowledgment

We thank CNPq and FAPERGS for supporting this work.

References

[1] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *2011 International Reliability Physics Symposium*, April 2011, pp. 5B.4.1–5B.4.7.

[2] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, Apr 1997.

[3] J. Carreira, H. Madeira, J. G. Silva *et al.*, "Xception: Software fault injection and monitoring in processor functional units," *Dependable Computing and Fault Tolerant Systems*, vol. 10, pp. 245–266, 1998.

[4] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, Dec 2003, pp. 29–40.

[5] X. Fu, T. Li, and J. A. B. Fortes, "Sim-soda: A unified framework for architectural level software reliability analysis," 2006.

[6] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Architecture-level soft error analysis: Examining the limits of common assumptions," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 266–275.

[7] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ace analysis reliability estimates using fault-injection," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 460–469. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250719

[8] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *International Conference on Dependable Systems and Networks, 2004*, June 2004, pp. 61–70.

[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)*, Oct 2015, pp. 211–214.

[10] E. Carlisle, N. Wulf, J. MacKinnon, and A. George, "Drseus: A dynamic robust single-event upset simulator," in *2016 IEEE Aerospace Conference*, March 2016, pp. 1–11.

[11] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos, "Differential fault injection on microarchitectural simulators," in *2015 IEEE International Symposium on Workload Characterization*, Oct 2015, pp. 172–182.

[12] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. D. Carlo, "Rt level vs. microarchitecture-level reliability assessment: Case study on arm(r) cortex(r)-a9 cpu," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2017, pp. 117–120.

[13] C. Celio, D. A. Patterson, and K. Asanović, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, Jun 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[14] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[15] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 69–78.