A Fast and Scalable Fault Injection Framework to Evaluate Multi/Many-core Soft Error Reliability

Felipe Rosa, Fernanda Kastensmidt, Ricardo Reis
UFRGS - Instituto de Informatica - PGMicro/PPGC
Av. Bento Goncalves 9500 Porto Alegre, RS - Brazil
Email: {frdarosa, fglima, reis}@inf.ufrgs.br

Luciano Ost
Department of Engineering - University of Leicester
University Road, Leicester, LE1 7RH, UK
Email: luciano.ost@leicester.ac.uk

***Abstract*—Increasing chip power densities allied to the continuous technology shrink is making emerging multiprocessor embedded systems more vulnerable to soft errors. Due the high cost and design time inherent to board-based fault injection approaches, more appropriate and efficient simulation-based fault injection frameworks become crucial to guarantee the adequate design exploration support at early design phase. In this scenario, this paper proposes a fast and flexible fault injector framework, called OVPSim-FIM, which supports parallel simulation to boost up the fault injection process. Aiming at validating OVPSim-FIM, several fault injection campaigns were performed in ARM processors, considering a market leading RTOS and benchmarks with up to 10 billions of object code instructions. Results have shown that OVPSim-FIM enables to inject faults at speed of up to 10,000 MIPS, depending on the processor and the benchmark profile, enabling to identify erros and exceptions according to different criteria and classifications.**

I. Introduction

Integrating multiple commercial Off-The-Shelf (COTS) processors in the same system is now commonplace in both embedded and high performance computing (HPC) domains [1]. Such systems aim to perform complex application workloads (i.e. billions of object code instructions), which will continue growing in diverse fields (e.g. aerospace, automotive, etc) [2]. To meet the system performance, processors of these systems may be required to operate in aggressive clock frequencies (i.e gigahertz). The high frequency operation and the continuous technology shrink are making underlying systems more susceptible to soft errors, such as the ones caused by radiation effects [3]. Soft errors or Single Event Effects (SEE) induced by neutron may cause critical failures on system behaviour, which can lead to financial or human life losses [4]. In this regard, the occurrence of SEEs has been considered as a major concern in memory cells and processors working at ground level.

Fault injection techniques are widely used to assess soft errors of embedded and HPC systems under fault campaigns at design time [5]. Aiming at accelerating the fault injection evaluation of such complex systems, simulation-based fault injection approaches are proposed to enable complex soft error resilience analysis regarding to different system configurations at acceptable time. Simulation-based fault injection frameworks allow early evaluation of the system reliability when only system components models are available. Although, simulations performed either at register or at gate level provide more accurate results, there are two main issues that reduce their relevance for performing fault injection campaigns in complex multi/many-core systems. Commercial processors are rarely available to users in register or gate-level description and required simulation time is extremely high even for relatively small processors, making the investigation of systems composed of more complex processors impractical.

More recently, researchers have been proposed to extend virtual platform simulators aiming to enable fault injection analysis at early design phases. Authors in [6] present the Relyzer, a hybrid simulation framework for SPARC core using Simics [7] and gem5 [8] simulators coupled with a pruning technique to reduce injected faults. In this work, a 200-cores cluster was employed and approximately 11 days were required to inject around 32 million faults, resulting in an average of 33 injected faults per second. In [9], 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 1 fault per second. Most reviewed approaches consider either small scenarios or a single target processor or specific ISA [7]. Further, such works typically report best-case simulation performances of 2-3 MIPS, allowing 33 fault injections per second considering a supercomputer [6].

Different from the above works, this paper proposes a fault injector framework, called OVPSim-FIM, developed on the basis of OVPSim [10] aiming to speed up the analysis of complex and large-scale systems composed of hundreds of commercial processors under neutron-induced faults. OVPSim-FIM can be used to inject faults into: memory area, CPU registers, and interconnection infrastructure. To validate the proposed framework, several fault injection campaigns were performed in ARM Cortex-A9, A15 and M4F. Fault injection reports are obtained by performing a trustworthy number of fault injections (at least 100,000 faults per campaign) considering a market leading RTOS and several well-known benchmarks. Another contribution of promoted framework relies on the possibility of performing parallel fault injection campaigns, defined according to the number of available host-CPUs. Underlying feature enables to simulate up to hundreds fault injections per second, considering normal desktop PCs (i.e. quad-core with 8GB of memory).

II. Proposed Fault Injector Framework

Figure 1 shows the main components of OVPSim-FIM: fault model library, FIM, platform model and simulation infrastructure. The fault model library (FML) combines fault injection techniques, which may be used to evaluate the dependability of single or multi-core platforms composed of three main components: memory, processor and interconnection. In this regard, designers can select the most appropriate fault injection (FI) technique for each set of platform components (e.g. processors, busses, routers or memory) according to their needs. FI models contain information about the behavior of the faults (e.g. bit location and fault injection time), which shall be performed by the fault injector module (FIM). Proposed FIM incorporates four main mechanisms: (i) fault configuration, (ii) fault monitor, (iii) error detection, and (iv) exception handler.

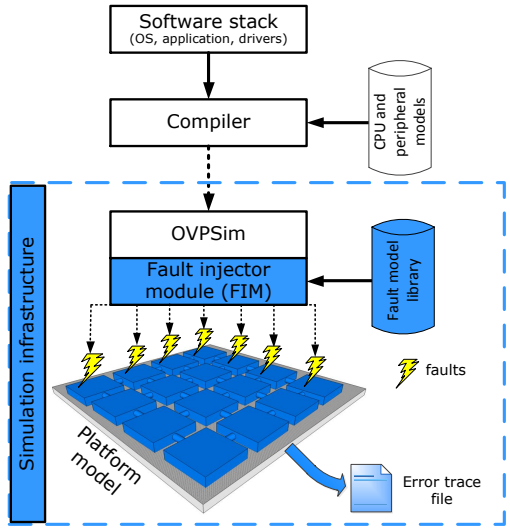

Fig. 1. Proposed fault injection framework organization, where white boxes represent OVPSim regular framework components and proposed FIM modules and associated functions are indicated by blue boxes.

To set the FIM, users must use the appropriate flags to inform target components/locations (e.g. processor, memory regions, etc) as well as the injection technique for each campaign. Users can either set up the *fault configuration mechanism* using flags or employ a fault description input file, which provides pre-defined parameters (e.g. number of faults, etc) for each fault campaign. The default fault injection configuration (e.g. bit location, injection time) relies on a random uniform function, which is a well-accepted fault injection technique since it covers the majority of possible faults on a system at a low computation cost [11]. The *fault monitor function* is used to check the system behavior and the occurrence of errors in the presence of faults. In turn, the *error detection* verifies the platform components (e.g. CPU control registers) context at simulation end in order to detect arising errors or consolidate exceptions captured by the *exception handler*.

OVPSim-FIM was developed according to OVPSim API environment, which has several functions defined as callbacks. Callbacks enable to access, to modify, and to control platform components during the simulation. For instance, in our implementation, when a predefined events occur (e.g. access to a give memory area) a callback function is used to stop the simulation and to trigger the fault monitor component execution.

A. Fault Injection Simulation Flow

Proposed fault injection flow comprises five phases, as illustrated in Figure 2. In the *golden execution* (phase 1), each application is executed in the original OVPSim distribution to verify its correctness and to extract essential information like instruction count and memory map. Resulting instruction count and memory map are used to create the final error report (phase 5), which provides fault injection positions and detected errors.

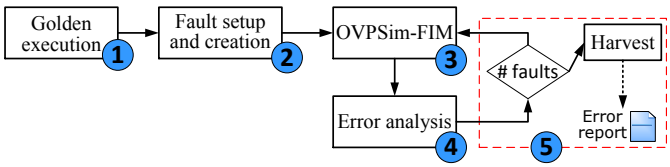

Fig. 2. Five phases of proposed fault injection flow using OVPSim-FIM.

The second phase involves fault configuration and fault list creation. As OVPSim, promoted framework relies on just-in-time compilation technology, which enables high simulation performance at the cost of limited accuracy. As consequence, one complete instruction is the minimum granularity achieved, independent of real cycle count. In this regard, the number of performed instructions is assumed as temporal reference in this work. Supposing the default fault injection configuration, injection time is defined randomly based on a number between one and the final instruction count extracted in phase 1. Following the principal, fault locations (e.g. registers, memory address) are also defined randomly. Faults are characterized as single bit-flips that can be injected in internal components like CPU registers (e.g. R0, PC, fps). To guarantee a single bit-flip injection, a bit pattern mask is generated for each fault. For instance, considering a 32 bit processor like adopted Cortex A-15, all 32 bits are set to 1 except the bit that will be flipped. Thus, to change the second bit of a 32 bits register, the 0xFFFFFFFD mask must be defined. As the previous steps, the pattern generation is also randomly. For this purpose, a 0x00000001 integer value is shifted up to 32 times to invert all bits of this integer. In the third phase, the fault monitor verifies the number of executed instructions in order to inject a fault. At this moment, the targeted component position (e.g. register, memory address) is accessed and related value is used with the previous generated mask pattern (phase 2) in an exclusive OR operation (XOR), inverting the collected value.

In phase 4 (*error analysis*), each application behavior running under fault injection is compared to the golden standard run in order to detect possible arising errors. OVPSim-FIM generates a error report according to Cho's classification [5]:

(i) *Vanished* - no fault traces are left; (ii) *Output Not Affected* (ONA) - the resulting memory is not modified, however, one or more remaining bits of the architectural state are incorrect; (iii) Output Mismatch (OMM) - the application terminates without any error but the resulting memory is affected; (iv) *Unexpected Termination* (UT) - the application terminates abnormally with an error indication; and (v) *Hang* - the application does not finish and it is preempted using a timeout.

B. Simulation Infrastructure

During the simulation, injected faults can generate different processor exceptions (e.g. write/read private memory regions), which can suspend the simulation. To overcome such restriction, a C-module of our simulation infrastructure was built to deal with such unexpected events at runtime in order to keep the simulation running until it reaches the end. Promoted simulation infrastructure enables to access and to deal at runtime with different exceptions individually, allowing to continue the execution of a faulty processor. In this context, the OVPSim-FIM resumes the simulation for all unfinished components; waiting until all processors finishes either by own means (i.e. reaching an exit or break instruction) or triggering another event. Note, our main goal is to capture arising errors, not to treat or to mitigate them, however proposed framework may be extended to support both process.

Proposed infrastructure includes two features to boost up the fault injection simulation: checkpoint and an independent parallel simulation engine. *Checkpoint* technique consists in collecting platform components context during the gold execution (phase 1) in order to restore the appropriate context later during the fault injection campaign, reducing the amount of re-executed code. Such technique is constructed using OVPSim save and restore functions, which allow restoring processor and memory context. Restored contexts are executed identically to an unmodified execution (i.e. complete simulation). For this purpose, during gold execution the application context, covering processor and memory models, is stored periodically according to a pre-defined number of instructions, which determines the interval between the checkpoints. The user can specify this interval or assign a number of checkpoints to the target application.

To benefit from host's processing capacity, a simulation infrastructure instrumentation is proposed to manage parallel host-cores resources utilization. Proposed simulation infrastructure comprises a set of C-based functions (e.g. management) and bash scripts developed according to OVPSim guidelines. Our simulation infrastructure was designed to be modular and highly configurable, however we advocate that the most suitable setup consider allocating one platform model per available host-core (default operation mode). Each platform model runs independently from fault injection (phase 1) to error report generation (phase 5). For instance, considering a quad-core processor machine, it is possible to execute 4 platform models injecting 1,000 faults each, totalizing 4,000 fault injections in parallel. In the present work, we adopt a platform model composed of several nodes interconnected by an OVP-based network-on-chip model, which was incorporated in the original OVPSim components library. Each platform node is composed of a single processor connected to the local memory and router by a bus. Due the modeling flexibility, different fault injection techniques can be employed to evaluate both homogeneous and heterogeneous processor platforms.

III. Experimental Setup and Results

Experimental setup: all fault injection simulations were performed in a Quad-core Intel(R) Core(TM) i7-4790K CPU (32 GB DDR3 RAM machine). Since OVPSim uses the target CPUs binary code to perform emulation on a host machine, all simulation scenarios were executed multiple times and only worst cases are reported here.

A. Case Studies Considering ARM Processors

In order to demonstrate the OVPSim-FIM fault injection capabilities, the proposed fault injection framework is used for simulating the effect of faults on 1000-node system platforms. Fault analyses are obtained by injecting faults (i.e. bit-flips) in general-purpose registers of three ARM processor models: Cortex-A9, Cortex-A15, and Cortex-M4F, executing14 benchmarks. During fault injection campaigns, errors are classified according to Cho's proposal [5] (Figure 3). Thousands of faults are injected in each ARM processor register (i.e. PC, SP and other 12 registers) in a random order, in order to estimate the percentage of errors that are not masked by each benchmark. The mean error percentage for all 42 scenarios is 60.1%, 60.2%, and 74.4%, respectively, for Cortex-A9, Cortex-A15, and Cortex-M4F, which executes a FreeRTOS. Cortex-M4F running the FreeRTOS presents a SDC rate of 58% several times higher than other two Cortex-A architectures, which execute only bare-metal applications, resulting in a SDC rate of only 2.4%. In contrast, both Cortex-A9 and A15 have a higher rate of memory output and incorrect processor state, 40% against less than 1% for M4F.

B. OVPSim-FIM Performance

This experiment evaluates OVPSim-FIM speedup when compared to the original OVPSim distribution, which uses one physical core, even in multicore processors as Intel-I7. To provide relevant metrics, FreeRTOS kernel version V.7.4.21 was ported to the ARM Cortex-M4F model available in OVPSim. The Adpcm, Fibonacci and PeekSpeed benchmarks are chosen due their workload profiles, i.e. number of instructions of each one, number of generated exceptions during simulation, etc.

Figure 4 compares developed simulation performance techniques using the original OVPSim as reference (white bars). The second bar shows the simulation performance of OVPSim-FIM when executed in one single host-core without checkpoint technique. The simulation overhead is 2.3%, 5.2%, and 12.6% for Adpcm, Fibonacci and PeekSpeed (baremetal), respectively, when compared to the original OVPSim distribution. Note, the more instructions the less simulation overhead.

The checkpoint technique achieves up to ten thousand MIPS when comparing with the simple OVPSim-FIM implementation. The PeakSpeed provides the smallest gain (only 3%).

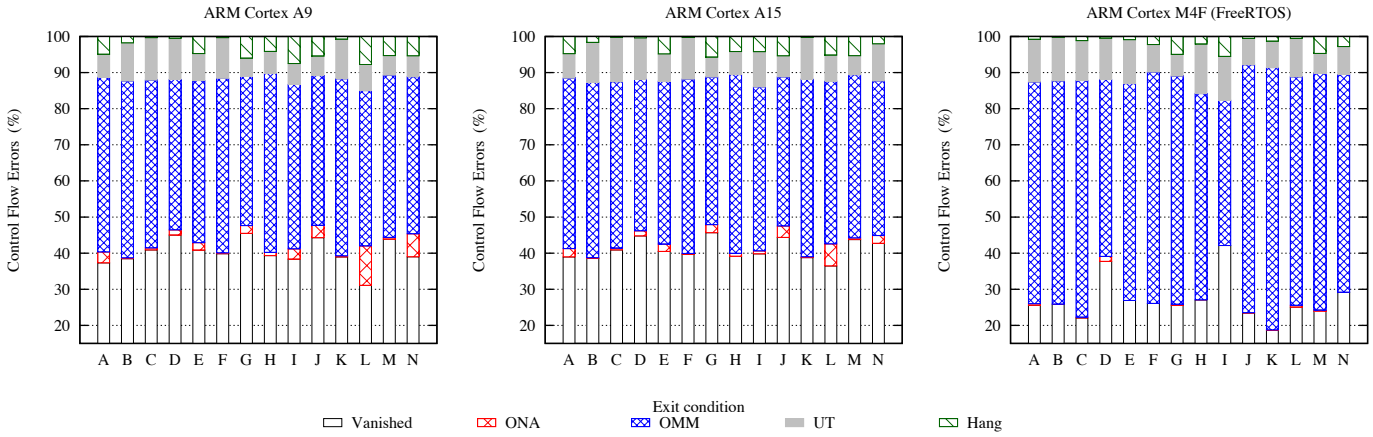

Fig. 3. Error Percentage for Cortex-A9, A15, and M4F, executing Adpcm (A), Blowfish (B), Bubble Sort (C), Edn (D), Factorial (E), Fdct (F), Fibonacci (G), Hanoi Tower(H), Harmonic Calculations (I), Insert Sort (J), Jfdctint (K), mdc (L), PeakSpeed (M), and Integer Matrix Multiplication (N).

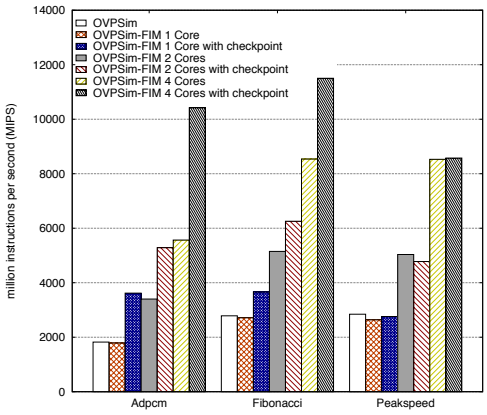

Fig. 4. Comparison between OVPSim-FIM simulation performance techniques, considering 3 benchmarks executing onto a 1000-node platform model, where each core executes one instance of FreeRTOS and a benchmark.

In this case, the overhead for restoring processor and memory context higher than executing the small number of instructions of this benchmark (around 118 millions). Nevertheless, larger and more complex applications like Adpcm and Fibonacci benefit from this technique, saving a considerable simulation time. Underlying results demonstrate the increased speedup of the proposed approach when compared to original OVPSim distribution, promoting large error/fault-oriented analysis as required by emergent multi/many-core systems.

IV. Conclusion

This paper presented two major contributions: first is targeting the general problem of accelerating the fault injection and analysis of complex and large-scale systems composed of multi/many-core processors; second is the focusing on the specific problem of evaluating commercial processors under soft errors scenarios. In light of that, we proposed the OVPSim-FIM fault injection framework that unifies design flexibility, fast simulation and high injection rate in one single simulator. The effectiveness of OVPSim-FIM was evaluated according to more than hundreds simulation scenarios, considering FreeRTOS, well-known benchmarks, and different ARM Cortex family processors. Future works includes the use of this framework to compare and evaluate different operating systems, multiple processor embedded and HPC systems, fault tolerant software-based applications.

References

[1] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with commodity cpus: Are mobile socs ready for hpc?" in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 40:1–40:12.

[2] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer*, vol. 42, no. 4, pp. 42–52, Apr. 2009.

[3] N. Seifert, "Radiation-induced Soft Errors: A Chip-level Modeling Perspective," *Found. Trends Electron. Des. Autom.*, vol. 4, no. 2-3, pp. 99–221, Feb. 2010.

[4] "Toyota Case: Single Bit Flip That Killed | EE Times." [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1319903

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

[6] 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*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 123–134.

[7] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Ha allberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

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

[10] Imperas, "Open Virtual Platforms (OVP)," 2014. [Online]. Available: http://www.ovpworld.org/

[11] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: Probabilistic Soft Error Reliability on the Cheap," in *Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, 2010, pp. 385–396.