Received: 12 August 2020 | Revised: 30 November 2020 | Accepted: 9 December 2020

DOI: 10.1049/cdt2.12017

ORIGINAL RESEARCH PAPER

Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator

Geancarlo Abich[1] | Rafael Garibotti[2] | Vitor Bandeira[1] | Felipe da Rosa[1] | Jonas Gava[1] | Felipe Bortolon[1] | Guilherme Medeiros[1] | Fernando G. Moraes[2] | Ricardo Reis[1] | Luciano Ost[3]

[1]School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

[2]PGMicro/PPGC, Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

[3]Wolfson School, Loughborough University, Loughborough, UK

Correspondence

Luciano Ost, Loughborough University, Epinal Way Loughborough Leicestershire LE11 3TU, UK.
Email: l.ost@lboro.ac.uk

Abstract

Soft error resilience has become an essential design metric in electronic computing systems as advanced technology nodes have become less robust to high-charged particle effects. Designers, therefore, should be able to assess this metric considering several software stack components running on top of commercial processors, early in the design phase. With this in mind, researchers are using virtual platform (VP) frameworks to assess this metric due to their flexibility and high simulation performance. In this regard, herein, this goal is achieved by analysing the soft error consistency of a just-in-time fault injection simulator (OVPsim-FIM) against fault injection campaigns conducted with event-driven simulators (i.e. more realistic and accurate platforms) considering single and multicore processor architectures. Reference single-core fault injection campaigns are performed on RTL descriptions of Arm Cortex-M0 and M3 processors, while gem5 simulator is used to multicore Arm Cortex-A9 scenarios. Campaigns consider different open-source and commercial compilers as well as real software stacks including FreeRTOS/Linux kernels and 52 applications. Results show that OVPsim-FIM is more than 1000× faster than cycle-accurate simulators and up to 312× faster than event-driven simulators, while preserving the soft error analysis accuracy (i.e. mismatch below to 10%) for single and multicore processors.

1 | INTRODUCTION

Industrial market leaders in automotive, medical, consumer electronics, and high-performance computing (HPC) sectors employ state-of-the-art processors and graphics processing units (GPUs) to fulfil the high computational demand of applications. Applications running on such electronic computer systems differ not only in performance but also in terms of security, reliability and power requirements. In the reliability aspect, the soft error resilience is emerging as a key design metric due to the increasing susceptibility of electronic computer systems to the occurrence of soft errors caused by radiation effects [1]. According to Baumann et al. [1], the collision of a sub-atomic particles (e.g. alpha particles and high-energy particles) induces a single event transient (SET) by generating secondary particles capable of ionizing the n-p junctions of sensitive transistors causing a voltage charge or discharge in the stroke node. Therefore, high-energy neutrons are becoming the primary source of soft error at ground-level surpassing the alpha particles [1, 2]. The occurrence of soft errors or Single Event Effects (SEEs) may cause critical failures on system behaviour, which may lead to financial or human life losses as already reported in [3].

The software and hardware complexity of such systems impose exploration challenges, including (*i*) conduct a large number of fault injection (FI) campaigns within a reasonable time, (*ii*) provide engineers with detailed observation of a system's behaviour in the presence of faults, and (*iii*) identify relationships or associations between application characteristics and specific platform parameters in large data sets resulting from the fault campaigns. Fault injection techniques are widely used to assess soft errors of embedded and high-performance computing (HPC) systems under fault campaigns at the design time [4]. Simulation tools normally inject faults using a processor design model. Some simulators are based on high level behavioural models and some based on hardware description language level (HDL) or gate level models. Although, HDL level and gate level models are more accurate than high behavioural models, there are two main problems when using HDL or gate level descriptions: commercial processors are rarely available to users and the necessary time to simulate detailed commercial multicore processors is extremely high, making the soft error analysis of emerging multicore processors impractical [5].

The resulting scenario calls for faster and more efficient means to assess the soft error resilience of complex systems with the minimal overhead in time-to-market. Virtual Platform (VP) frameworks have gained popularity over the years not only in academia but also in many industrial sectors, due to their design flexibility, debugging and simulation performance capacities. In this sense, works incorporated fault injection capability into VP frameworks [6,7], enabling the analysis of complex software stacks and processor architectures at early design phases. While the gain in simulation speed is trivially observed in VP simulators based on just-in-time (JIT) dynamic binary translation, the soft error assessment consistency of underlying fault injection frameworks remains unclear. An initial effort to investigate the soft error consistency of VP simulators has been conducted in [5,8,9]. Common findings of these works is that higher level FI approaches provide more flexibility and simulation performance at the cost of accuracy, mostly resulting from the lack of microarchitecture and timing modelling aspects.

In this direction, the *main contribution* of this paper is an in-depth and statistical significance soft error consistency evaluation of a JIT-based fault injection framework (OVPsim-FIM [7]). This work aims to further contribute to the use of such frameworks in the early design phase of safety-critical applications by demonstrating that software engineers can rely on soft error results produced from JIT-based frameworks throughout a comprehensive and thorough study with more than 12 million fault injections. In this sense, *the other contributions* of this article are related to assessing the impact of (*i*) standard open-source (Clang and GCC) and commercial compilers (Arm) on the soft error reliability; and (*ii*) virtual platform parameters that directly impact on soft error assessment. This assessment considers two reference models: real single-core commercial processors at the register-transfer level (RTL), and a multicore processor model available on the cycle-accurate gem5 simulator.

The sequence of this article is organised as follows. Section 2 discusses related works on fault injection approaches. Section 3 introduces the fault injection engine, presents the adopted fault classification and details the applied statistical model. Next, Section 4 explores the single-core soft error consistency, while Section 5 evaluates the multicore soft error consistency of OVPsim-FIM. Finally, Section 6 points out the conclusions.

2 | RELATED WORKS

Fault injection simulation frameworks are gaining utmost importance due to their efficiency to obtain prominent results on the soft error resilience of different systems, early in the design phase. These FI frameworks can be divided into two classes: (*i*) those focussed on the accuracy of the results and (*ii*) those that deal with highly complex systems or have little time to explore the project reliability. Simulation-based soft error analysis at RTL or gate levels are examples of the first class. Within this category, Mansour and Velazco [10] presented a method called Direct Fault Injection (DFI) to emulate the consequences of a single event upset (SEU) occurring in the processor's memory cells. However, this approach requires modification of the circuit architecture, which is not easily applied anywhere. Abbasitabar et al. [11] also investigated the susceptibility of the LEON3 processor in RTL, but their approach relies on the Modelsim simulator to inject faults without explicitly changing the reference design. Although both approaches produce accurate results, they are restricted to relatively small systems and experimental fault campaigns due to their low simulation performance.

Researchers started investigating other approaches, such as the use of virtual platforms, aiming to boost the soft error analysis of complex systems comprising not only real software stacks but also instruction set architectures (ISAs) and state-of-the-art processors [6–8]. While Parasyris et al. [6] introduced GemFI, a fault injection tool based on the cycle-accurate full-system model of the gem5 simulator [12]. Rosa et al. [7] presented a fault injection framework based on another virtual platform, the OVPSim [13], which has the advantage of supporting parallel simulation to boost up the fault injection process. Such VP-based FI frameworks are used to investigate different software stacks configurations, such as standard parallelisation libraries [14] and compiler optimization flags [15–17], and their impact on soft error reliability. For instance, authors in [15–17] investigate the impact of GCC compilation flags (e.g. O0, O3, etc.) on the application behaviour under fault influence. These works consider either simple (i.e. in-house and bare-metal applications) or small scenarios, where only a single processor is considered. Modern compilers have specific characteristics, which directly impact on applications performance, power-efficiency and reliability [18]. Taking a step forward in compiler assessment, Serrano-Cases et al. [19] proposed a method to find out the best combination of compiler optimizations and parameters to improve the fault tolerance of applications. In this regard, our work is complementary to [19] that evaluated x86 processors, as it aims to find out which is the best compiler combined with an optimization flag for Arm processors.

Aiming to demonstrate the discrepancies between fault injections conducted at different levels, researchers have conducted different assessments. For instance in [20], the authors compare soft error results of an intermediate code-level fault injection IR-level tool (LLFI [21]) and an assembly-level fault injector, considering a X86 processor. Results show similar silent data corruption (SDC) values, but the mismatches for crash results reach almost up to 40% in some cases. The main reason for this behaviour is that the instructions in LLVM IR can be translated into more than one assembly instruction and vice-versa. Also, some instructions are handled differently in the two code levels (e.g. instructions involving pointers). Cho et al. [4] evaluate the accuracy trade-offs associated with a variety of high-level fault injection techniques (i.e. RTL) comparing to a flip-flop-level baseline. Similarly, Schirmeier et al. [22] apply gate-, flip-flop- and ISA-level (i.e. register file) FI techniques to evaluate error-rate discrepancies considering the gate-level FI as reference. Authors in [4] use geometric means to show the mismatch between the FI levels. Although useful, the authors mention that the adopted metric may not capture how mismatch levels vary across various applications. To improve the soft error assessment accuracy analysis, authors in [22] use a ranked correlation to evaluate the mismatch between the FI approaches considering the extrapolated absolute failure count (EAFC) [23] normalized according to the gate-level FI. Such an approach ranks the results from FI techniques of each application considering the rank shuffling to evaluate the mismatch between the FI approaches. Authors demonstrate that even with discrepancies in such an approach, ISA-level FI approaches are sufficient to evaluate the soft error reliability of low-resource constraint processors (e.g. Cortex-M0). However, the EAFC metric only considers the occurrence of SDCs, which might lead to an inadequate mismatch assessment between different level of FIs since not all fault classifications are taken into account.

Kaliorakis et al. [8] were the first ones to be concerned with the accuracy of such FI frameworks based on virtual platforms, comparing FI implementations for gem5 [12] and MARSS [24]. In the same sense, Chatzidimitriou et al. [9] proposed to analyse the soft error rate accuracy of a gem5-based FI framework against results obtained from a neutron beam experiment, considering an Arm Cortex-A9 processor and 13 benchmarks. The proposed work is complementary to [9] because it considers other parameters (e.g. cross-compilers) and the soft error consistency analysis focus on a JIT-based virtual platform. An initial effort to evaluate the soft error assessment consistency of a JIT-based FI framework is described in [5]. In this work, a Fault Injection Module (FIM) was integrated into gem5 and OVPsim. Results considering several fault injection campaigns were compared, and a mismatch of up to 20% is reported. In further experiments, the Authors achieved a lower worst-case mismatch of 12% by reducing the simulation granularity of OVPsim, that is the number of instructions executed per simulation cycle. Similar to Kaliorakis et al. [8], conducted experiments aim to evaluate and provide insights on how to improve the accuracy of FI frameworks based on virtual platforms. Our work differs from [5], concerning the investigation of multicore systems, is that we introduce the discussion on the different cross-compilers and parallel programing models (i.e. Serial, MPI, and OpenMP), considering four cores and the NAS Parallel Benchmarks (NPB) [25].

This work is distinguished from those found in the literature by making an extensive and statistical significance soft error consistency assessment of a JIT-based fault injection framework. To the best of our knowledge, this work is *the first* to cover so many aspects together, such as single and multicore processors (i.e. Arm Cortex-M0, Cortex-M3, and Cortex-A9); three parallel programing models (i.e. Serial, MPI, and OpenMP); operating system (i.e. FreeRTOS, Linux kernel); five sets of compilers and versions (i.e. GCC 4.9, GCC 7.2, Clang 6, Arm 6.10 and Arm 5.06); optimization flags (i.e. O0, O1, O2, O3, Os and Ofast); and more than 50 applications taken from the NPB [25], Rodinia [26], and the Mälardalen WCET benchmark [27]. This range of parameters led to more than 12.7 million fault injections, bringing an excellent confidence level to the results.

3 | FAULT INJECTION METHODOLOGY

This section details the adopted fault injection approaches and their different level of accuracy (Section 3.1). Integrated FIMs (Fault Injection Modules) combine fault injection techniques, which may be used to evaluate the soft error resilience of single or multicore systems. In this sense, each fault injection module is detailed to show its usefulness, that is, RTL (Section 3.1.1), gem5 (Section 3.1.2), and OVP (Section 3.1.3). Next, Section 3.2 details the adopted fault classification, which is implemented in the three FIMs so that the results are automatically classified. Finally, Section 3.3 presents the assessment metrics used in this work, which guarantees the statistical significance of the results and subsequent conclusions of the research.

3.1 | Fault injection frameworks

This section first describes two event-driven fault injection frameworks based on RTL (Section 3.1.1) and gem5 (Section 3.1.2), which are used in the present work as *references* for the assessment of single and multicore systems, respectively. Then, the main functionalities of OVPsim-FIM are presented in (Section 3.1.3).

3.1.1 | RTL —Fault injection module

This section depicts the developed fault injection module for RTL descriptions, similar to that presented in [11,28]. The main distinction between FIMs is the moment in which the fault is injected. RTL-FIM executes under a discrete event model of computation (e.g. Questa Advanced Simulator), enabling the injection of faults at any or even within half a clock cycle. Therefore, inserted faults may affect the behaviour of both the current and the next instructions. Developed RTL fault injection module explores built-in simulator commands and its observability capability to control and monitor the internal signal of a given processor without requiring any changes in its description.

This approach assumes that the fault injection campaigns comprise the four phases illustrated in Figure 1: (*1*) *Golden Reference Model*; (*2*) *Fault Injection Setup*; (*3*) *Fault Injection Simulation*; (*4*) *Fault Injection Analysis*. In the first phase, the FIM executes the target system to extract its behaviour under ideal circumstances (i.e. no presence of faults). To improve the performance of the RTL fault injection campaign, this step generates checkpoints from simulation time slices. Then, in the Fault Injection Setup phase, the engine defines the fault configuration, which consists iof its location (e.g. register, memory address), position (e.g. register bit) and its insertion time. The 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 [29]. In the Fault Injection Simulation phase, the FIM loads the checkpoint, executes the target system architecture in the presence of the configured faults (i.e. flipped bits), and extracts its behaviour. Throughout the Fault Injection Simulation, the developed engine uses available interrupt signals to detect any unexpected activity. Finally, in the last phase, the FIM compares the results against its golden reference data to automatically classify the occurred faults.

3.1.2 | gem5—Fault injection module

gem5 [12] was selected among the available cycle-accurate virtual platform simulators due to its open and free availability as well as its support for the Arm processor architectures with four CPU models, which differ in speed/accuracy trade-offs. This work employs the Out-Of-Order (O3) model, which is the most detailed one, as reference for our experiments. Gem5 also supports a rich set of component models, including processor cores, memories, caches, and interconnections. It targets microarchitecture explorations, which incurs in substantial simulation overheads due to the number of modelled aspects. Typically, it reports best-case simulation performances of up to two to three million of instructions per second (MIPS).

Additionally, gem5 is a well-known simulator used in many research projects underlying the soft error assessment systems [6,8,9,14]. Our gem5-based FIM follows the four-phase fault injection flow illustrated in Figure 1. Although the phase split is the same for the three FIMs, each has its own implementation. The first difference with RTL is in the virtual platform setup (VP Setup). Here, the application, the kernel, and the configuration of the target architecture are compiled to simulate together in the first phase of the flow. Another difference is how to inject a fault. While RTL relies on proprietary Questa Advanced Simulator commands such as *force -deposit*, gem5-FIM employs Python scripts to control the simulation flow and use C/C++ modules to model the microarchitectural components. This deployed fault injection approach minimises intrusion into the simulator's engines, allowing any researcher in possession of the original simulator to use, modify, or extend its functionality.

3.1.3 | OVPsim—Fault injection module

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. Among available JIT-based virtual platforms, OVPsim [13] distinguished by its rich number of component models, which includes 170 processor variants, memories, UARTs, among other components. The support of this rich set of features combined with its simulation performance justifies the adoption of OVPsim in this work.

Regarding the soft error assessment, the developed FI module emulates the occurrence of SEU by injecting one flipped bit in a single register or memory address during the execution of a given software stack. In this work, SEU target only storage elements due to its higher susceptibility to radiation events when compared to logic elements [30]. Furthermore, OVPsim-FIM is an instruction-accurate simulation engine, and thus faults can only occur between instructions. Consequently, injected faults can only influence the behaviour of the next instructions. This lack of accuracy will be decisive for the mismatches in relation to the references FI approaches presented in Sections 3.1.1 and 3.1.2.

The proposed FIM also relies on the four-phase fault injection flow shown in Figure 1. However, OVPsim-FIM has an improved fault injection infrastructure, which includes two simulation techniques to boost up the fault injection assessment: checkpoint and an independent parallel simulation engine. The checkpoint technique consists of collecting platform components context during the Golden Reference Model (phase 1) to restore the appropriate context later during the FI campaign, reducing the amount of re-executed code and, consequently, accelerating the simulation time. This technique is built using OVPSim's save and restore functions, which allow restoring the processor and memory context. The system context is periodically saved during a faultless execution and later restoring the appropriate context for each fault injection. During faultless execution, the developed FIM stores (i.e. according to a predefined instruction interval) the application

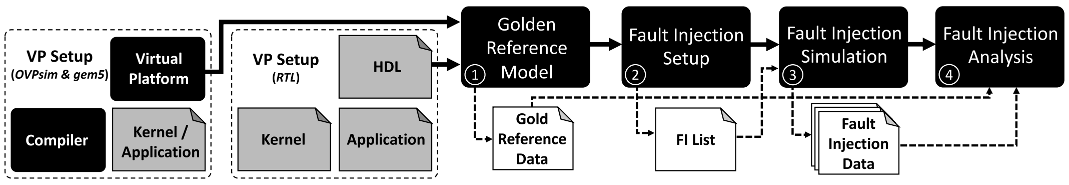


FIGURE 1 Fault injection campaign flow applicable to the three fault injection approaches, that is RTL, gem5 and OVPsim

context covering processor and memory models. At the fault campaign, each fault injection module identifies the closest checkpoint before the fault injection time to be restored. Additionally, the fault injection event trigger adjusts the injection time considering the fast for the number of forwarded instructions. The user can specify this interval or assign some checkpoints, and thus, the simulation infrastructure automatically estimates the interval between checkpoints.

In addition, the independent parallel simulation technique benefits from the host's processing capacity. The proposed simulation infrastructure comprises a set of C-based functions (e.g. management) and bash scripts developed according to the OVPSim guidelines. The goal is to allocate one platform model per available host-core, enabling the execution of multiple fault injection campaigns in parallel. For example, considering a quad-core processor machine, it is possible to run four platform models injecting 1000 faults each, reaching 4000 fault injections in parallel. This being one of the techniques used to be able to assess a huge number of FI scenarios in this work.

The fault injection process does not change the target application source, instead, the fault injector interrupts the simulator a single time to change a single bit in the application execution. The underlying interruption occurs a single time during a simulation, having an unnoticeable impact on the overall simulation time whenever compared to unmodified application execution.

3.2 | Fault classification

To characterise the target architecture behaviour in the presence of faults, the three fault injection modules execute the FI campaigns according to a statistical analysis (Section 3.3), aiming to achieve a reasonable confidence level so that additional campaigns do not disturb the result. The gathered results are classified according to Cho et al. [4], which defines five possible behaviours for a system in the presence of single-event upsets:

- *Vanish*: no fault traces are left in both memory and architectural state;
- *Output not Affected (ONA)*: the resulting memory is not modified, however, one or more remaining bits of the architectural state is incorrect;
- *Output Memory Mismatch (OMM)*: the application terminates without any error indication, and the resulting memory is affected;
- *Unexpected Termination (UT)*: the application terminates abnormally with an error indication;
- *Hang*: the application does not finish requiring a pre-emptive removal.

3.3 | Assessment metrics

One of the main concerns when assessing the reliability of a system is to develop a precise, well-covered and realistic approach. In this sense, this work sought to ensure that the number of fault injections has a statistical significance by applying the equations developed by Leveugle et al. [31]. Equation (1) shows the minimum number of campaigns needed to cover a certain confidence level with their respective margin of error. While the confidence level assures that if we repeat the experiments, the same results are obtained. Also, the margin of error indicates the percentage difference between the obtained results and the real value of the population.

$$n = \frac{N}{1 + e^2 \tilde{n} \frac{N-1}{t^2 \tilde{n} p \tilde{n} (1-p)}} \tag{1}$$

where: N is the initial population, p is the estimated probability of a failure (defined as 50%), e is the considered margin of error, and t is the minimum required confidence level.

Our initial population is the product of possible spatial and temporal fault injections, that is the location (e.g. register, memory address) and the position (e.g. register bit) at which the bit-flip will be applied to and its insertion time (e.g. a random clock cycle), respectively. An important characteristic of the Equation (1) is that when the initial population (N) is large, and its increase has little influence on the FI campaign size for a given margin of error and confidence level. For example, for a population greater than one million (N), if the chosen confidence level is 99% (in which the calculated t corresponding to 2.575,829,303,549), with a margin of error of 5% (e = 0.05), we must perform at least 664 fault injections to provide the necessary confidence in our results.

4 | SOFT ERROR CONSISTENCY ASSESSMENT FOR SINGLE-CORE PROCESSORS

This section aims to thoroughly assess the OVPsim-FIM soft error consistency when targeting single-core processors. So, this work considers two real commercial RTL processor descriptions from the Arm Cortex-M family: Cortex-M0 and Cortex-M3; both available under the Arm University Programme [32]. OVPsim-FIM natively supports both processor models, and the synthesis-ready netlist descriptions of the underlying processors were used to conduct the fault injection campaigns at the RTL level.

4.1 | Experimental setup

To provide trustworthy results, experiments consider more than 8.5 million fault injections using 26 applications and varying parameters such as target processor, software stack, and cross compiler with its optimization flags. Table 1 presents the proposed experimental setup used to measure the soft error assessment consistency of OVPsim-FIM with respect to the RTL approach.

Processors	**Arm Cortex-M0 and Cortex-M3**
Software Stack	Bare-metal and FreeRTOS
Benchmark Suite	Mälardalen WCET [27]
Compilers	GCC 4.9, GCC 7.2, Clang 6, Arm 6.10 and Arm 5.06
Optimization Flags	O0, O1, O2, O3, Os, Ofast, Ospace* and Otime*
Number of Compiler Sets	32
(with optimization flag)	
Number of Applications	26
Injections per Scenario	1000 (Section 4.3.1) and 10,000 (Section 4.3.2)
Total Fault Injections	208,000 (Section 4.3.1) and 8,320,000 (Section 4.3.2)

TABLE 1 RTL versus OVPsim Experimental Setup. The * means a sub-flag used in conjunction with other standard flags (e.g. O0, O1, O2, O3)

Selected applications from the Mälardalen WCET benchmark suit [27] include: Adpcm, Binary Search, Bit Manipulation, Blowfish, Bubble Sort, Counts, CRC, Data Compression, Dhrystone, Edn, Exponential Integral, Factorial, Fdct, Fibonacci, Hanoi Tower, HArmonic Calculations, Insert Sort, Jfdctint, Matrix Multiplication, MDC, PeakSpeed, Petri Net, Prime Numbers, Switch Cases, Ud, and Usqrt.

Results were performed on a Linux machine with a Quad-core Intel® Core™i7-7700K CPU and 32 GB DDR3 RAM memory. Fault analyses are obtained by injecting faults (i.e. bit-flips) into the processor's registers (i.e. R0-R15) in a random and uniformly distributed manner, which is widely accepted that circuits are affected [29]. The purpose is to analyse the parameters and Arm processors according to the architectural vulnerability factor (AVF) [29], that is a percentage estimate of errors that are not masked, considering the different applications.

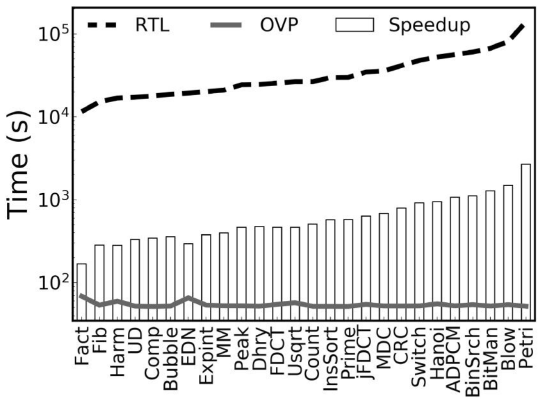


FIGURE 2 Speedup of OVPsim-FIM over the RTL-FIM, considering the Cortex-M0 executing 26 benchmarks in bare-metal

4.2 | FI simulation performance of OVPsim-FIM w.r.t. RTL

Although electronic hardware engineers usually describe their circuit designs using hardware description languages (HDLs) and fault injection at that level seems to be more straightforward due to the accuracy of the results, there are two main reasons for not using HDL models. First, commercial processors are rarely available to users in HDL descriptions. Second, the simulation time at this level is exceptionally high. This made the simulation speedup of fault injection campaigns one of the leading motivations found in the literature to use virtual platforms [5,8]. In this sense, our first experiment is to quantify the simulation speed for each of the adopted applications, as shown in Figure 2.

Figure 2 depicts a comparison between the simulation time required to execute a complete fault injection campaign in the RTL and OVPsim-FIM approaches. To make comparisons simpler and trustworthy, simulation time was extracted considering the Arm Cortex-M0 and bare-metal applications. Results show that OVPsim-FIM achieves a remarkable simulation performance, reaching a speedup greater than 1000× compared to the detailed fault injection approach conducted at RTL. Thus, if RTL is considered, thousands of simulations may take several months, which is not suitable to assess the soft error resilience of electronic computing systems. In this context, the utilization of a fault injection JIT-based virtual platform is promising since it can also be used for comparison among different processor models, ISAs and benchmarks considering complex OSs and large scenarios, as shown in Section 4.3.

4.3 | Soft error mismatch assessment

This section presents the results on the accuracy of the soft error resilience assessment in single-core processors, comparing OVPsim-FIM and RTL-FIM. Following subsections details each proposed experiments to provide a complete consistency analysis.

4.3.1 | Mismatch analysis considering processor architectures

Our first discussion is related to the Arm processor architectures, considering the ISA and the adopted software stacks.

This will provide us insights into which processor is more reliable and why. In this regard, Equation (2) is used to provide the mismatch between our accurate reference (i.e. RTL) and the OVPsim-FIM. The mismatch here is defined as the difference between results obtained for each application and the same fault class divided by the number of injected faults.

$$Mismatch = \frac{(RTL_{[\ app,\ class\ ]} - OVPsim_{[\ app,\ class\ ]})}{Number\ of\ Injected\ Faults} \qquad (2)$$

The first results highlight the effects of the Arm Cortex-M0 and Cortex-M3 architectures and their respective ISAs on soft error resilience. Although both processors are from the Cortex-M Family, they differ in terms of computer architectures (i.e. Von Neumann and Harvard respectively). Cortex-M3 has a larger ISA (i.e. entire thumb and thumb-2, 32-bit and 64-bit result multiplication, and 32-bit quotient division), while Cortex-M0 has a reduced ISA (i.e. most thumb, some thumb-2, and 32-bit result multiplication).

Different from Refs. [4,22], our evaluation considers an empirical data distribution that shows the minimum, maximum, and mean mismatch values with the interquartile ranges. Such data distribution enable us to show a detailed inter-benchmark mismatch variation considering all fault classifications and not only the EAFC [22] nor only the overall accuracy [4]. Figure 3 shows the mismatch distribution for each fault class, considering the impact of using different processors and software stacks. This means that each bar in Figure 3 represents 26,000 fault injections, with a confidence level of 99.8% and an error margin of 1%, according to the Equation (1). In addition, the experiments used the *GCC 4.9.3* compiler with *O0* optimization flag, which facilitates the reproducibility of the experiments by other researchers.

First, we see that the architectural difference between the two processors affects the system's reliability, producing different fault class behaviours, as shown in Figure 3. A major architectural difference is that the Arm Cortex-M0 has a reduced instruction set, which makes it to execute a larger set of instructions and to complete more complex operations than the Cortex-M3. For instance, while a 32-bit multiplication operation may vary from 1 up to 32 cycles in the Cortex-M0, the Cortex-M3 multiplier instruction needs a single cycle to complete the same operation.

Looking at Figure 3 globally, the OVPsim-FIM results differ more prominently, from the reference, in the occurrence of *ONA* and *UT* faults, whereas the mean is out of $\pm 5\%$ (i.e. a low mismatch reference). In this regard, the RTL approach is more likely to mask an injected fault targeting a general-purpose register (i.e. *Vanished*) or to propagate it to other registers (e.g. *ONA* and *UT*). In addition, the results also differ due to the simulation nature of each fault injection approach. In OVPsim-FIM, faults are injected between instructions, which can restrict their propagation and, consequently, reduce the cumulative effects of soft errors. But, the nature of discrete event simulators allows the injection into any clock cycle, which can affect the execution of multi-cycle instructions (e.g. a division execution can take 2 to 12 cycles on the Cortex-M3). In this case, the value of a register can change mid instruction, which may lead to either a masked fault, or an undefined processor state, or even a wrong application execution.

On the other hand, looking at specific points, we can see that the mismatch distribution between bare-metal scenarios revealed a higher occurrence of *ONA* (i.e. mean value indicated by (1) in Figure 3) for the Cortex-M0 on RTL approach, and a lower mismatch with Cortex-M3 (i.e. mean value highlighted by (2) in Figure 3). Also, results show that OVPsim-FIM presents lower mismatches of *Hang*, *OMM*, and *Vanish* faults for both processors, that is most mean values are in the $\pm 5\%$ range.

Regarding the operating system, when it is present, more functions (e.g. memory allocation, context switching) are available, increasing the software stack complexity. Results in Figure 3 (highlighted by (3) and (4)) show that the mismatch distribution from *ONA* faults is more disperse when using OVPsim-FIM, which affects the other fault types proportionally, that is the sum of the variations in the average mismatch between bare-metal and FreeRTOS is equal to zero. The collected results show that the mean values remain between00B1⁄4%, confirming that the inclusion of FreeRTOS did not affect the OVPsim-FIM soft error assessment accuracy.

FreeRTOS is a minimalist OS, that is a tiny footprint kernel with a lower overhead w.r.t. the application execution time. Thus, the probability of a fault striking the RTOS kernel

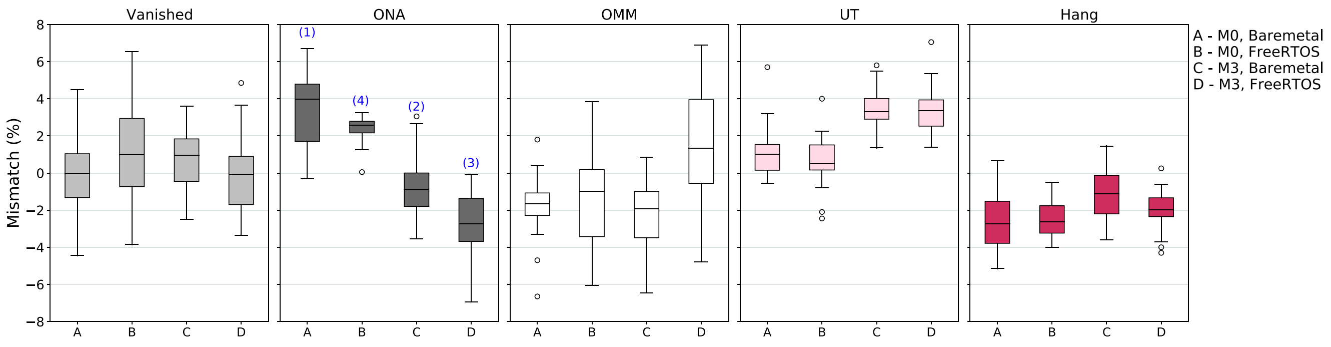


FIGURE 3 Boxplot of fault mismatch between OVPsim-FIM and RTL-FIM considering different architectures and software stacks. Outlier cases (represented by circles) are at least two standard deviations from the mean

functions is infinitesimal, mainly due to the low vulnerability window of the kernel w.r.t. to the execution overhead of complex applications. A similar behaviour is also found in more complex scenarios [14], including more robust kernels, such as a Linux, executing multithread applications. One of the effects, of running the FreeRTOS or any OS is the occasional context switch, which overwrites the registers potentially masking dormant faults in the registers. The application length impact on reliability depends on its behaviour. For example, some algorithms accumulate errors (e.g. matrix multiplication) while others fetch new data for each iteration (e.g. Video decoder).

4.3.2 | Mismatch analysis considering cross-compilers

Compilers play an important role due to their direct impact on applications performance, power-efficiency, and reliability [18], as they provide software engineers with a wide variety of optimization settings (i.e. flags), which can be used to either configure debugging and warning messages, or to achieve code optimization. In addition, industrial leaders employ different compilers in their projects. So, assessing the impact of these compilers on soft error reliability is vital to guarantee the success of their products. If, on the one hand, most of the work on compilation flags in the literature focuses on performance optimization [33], and into reducing memory usage and code size [34]. Few are those that assess the soft error reliability provided by compilers [15,17,19]. In this scenario, this section investigates the impact of widely adopted compilers and their optimization flags on the soft error reliability to find the most reliable set for Arm processors.

The long simulation time of the RTL approach restricts its use for the soft error assessment of multiple scenarios. For that reason, the evaluation considers only the Arm Cortex-M3 to investigate whether the nature of cross-compilers and optimization flags affect the accuracy of soft error results obtained with OVPsim-FIM. To handle this, five cross-compilers are considered:

- *GCC 4.9.3*: free-software, still widely used by legacy systems and applications;
- *GCC 7.2.1*: free-software, currently shipped with popular Linux distributions;
- *Clang 6.0.1*: free-software, which is an LLVM-based compiler;
- *Arm 5.06*: proprietary-compiler that is compatibility GCC;
- *Arm 6.10*: proprietary-compiler developed based on the LLVM compiler.

These compilers consider six optimization flags (i.e. *O0*, *O1*, *O2*, *O3*, *Os*, and *Ofast*), with the exception of the Arm 5.06 compiler, which has a different approach for *Os* and *Ofast*. In this case, the traditional flags *O0*, *O1*, *O2*, and *O3* are combined with other flags (i.e. *Ospace* and *Otime*), which are equivalent to either *Os* and *Ofast*, resulting in eight flag combinations.

Table 2 presents a mismatch percentage summary of the compiler sets (i.e., compiler with optimization flag) between the RTL and OVPsim FI modules. Considering only bare-metal applications, each compiler set comprises 26,000 fault injection campaigns, which means that our results have a confidence level of 99.8% and a margin of error of 1%, according to Equation (1). The results show that compilers have an impact on soft error resilience. However, the mismatch between the two approaches is very low, with means close to 2 for all compiler sets.

Regarding the mismatch distribution, worst-case scenarios maintain the absolute mismatch below 8%. Considering that the calculated margin of error is at 1%, and even the outlier values are very close between the two FI approaches, which means that OVPsim-FIM provides good reliability results being more than 1000× faster than RTL. But, comparing the compilers, the results of the 2 GCCs show that they have the highest number of outliers in worst-cases (see red values in Table 2). These outlier values may not yield an apparent correlation, but they present a difference in terms of executed instructions. For example, the GCC versions of the Dhrystone application execute 1.7× more instructions than the version generated by the Arm 5.06 compiler, which affects the application vulnerability window.

After presenting an overview of the compilers and optimization flags showing the accuracy of the results in relation to RTL, we increased fault injection campaigns to 10k for each compiler set and compared them using OVPsim-FIM. The goal is to find the most reliable compiler set, so the FI campaigns were increased to produce a more significant result in terms of confidence level and lower the margin of error. Thus, for each compiler set 260k FI campaigns are conducted, leading to a confidence level of 98%, and margin of error to 0.2%.

Figure 4 shows the soft errors related to the FI campaigns for the five considered compilers and their optimization flags using OVPsim-FIM. We consider AVF to be the percentage of the sum of all faults that could be analysed (i.e., *ONA*, *OMM*, *UT* and *Hang*). Thus, reliability is related to the percentage of *Vanishes* found. Among the evaluated compilers, the commercial *Arm 6.10* presents more occurrences of *Vanish*, indicating that it has a greater soft error resilience. From the open-source alternatives, *Clang* appears to be the best option due to two main reasons: (*i*) a higher number of vanishes was identified, and (*ii*) its adoption leads to a lower occurrence of *Hangs*. On the other hand, the GCC compilers followed the trend shown in Table 2 and continued to present the worst results in terms of soft error resilience, where no optimization flag passed the dashed red line shown in Figure 4a. To facilitate this view, Figure 4b is a zoom-in considering only *Vanish*. Although the newer GCC version uses more specific instructions of Arm Cortex-M3 ISA, the resulting

TABLE 2 Mean and worst-case absolute mismatch percentages between the RTL and OVPsim fault injection modules considering different cross-compilers and their optimization flags in Arm Cortex-M3. The * means a combined flag equivalent to *Ofast*

		Arm 5.06								Arm 6.10					
		O0	O0*	O1	O1*	O2	O2*	O3	O3*	O0	O1	O2	O3	Os	Ofast
Vanished	Mean	1.5	1.6	1.5	1.6	1.6	1.8	1.6	1.8	1.4	1.7	2.0	1.9	1.8	2.1
	Worst	4.2	6.1	4.7	4.2	5.6	5.1	4.9	5.9	7.5	4.9	5.4	5.9	5.2	5.1
ONA	Mean	1.1	1.1	1.1	1.4	1.5	1.6	1.3	1.6	1.8	2.9	1.6	1.2	1.7	1.3
	Worst	3.6	5.3	4.6	4.1	4.9	4.9	5.0	6.0	6.8	7.1	5.6	6.2	6.0	5.4
OMM	Mean	1.2	1.5	1.1	1.8	1.3	2.0	1.2	1.6	1.2	1.9	1.4	1.3	1.4	1.3
	Worst	4.0	4.5	3.8	5.5	3.1	5.2	3.2	5.4	4.8	5.5	6.4	3.5	3.3	5.1
UT	Mean	1.6	2.6	1.2	2.4	1.2	2.0	1.4	1.4	1.3	1.7	1.5	1.7	1.7	1.9
	Worst	6.1	5.9	5.2	6.6	5.7	5.6	5.2	3.1	3.4	4.3	4.2	3.9	4.5	4.5
Hang	Mean	1.3	1.6	1.4	1.6	1.4	2.2	1.7	1.9	2.0	1.5	1.7	1.5	1.7	1.6
	Worst	4.1	3.8	4.4	4.4	4.6	4.4	5.8	4.7	6.0	3.6	4.4	3.4	4.3	3.5

		Clang 6.0.1						GCC 4.9.3						GCC 7.2.1					
		O0	O1	O2	O3	Os	Ofast	O0	O1	O2	O3	Os	Ofast	O0	O1	O2	O3	Os	Ofast
Vanished	Mean	0.9	1.7	1.5	1.5	1.9	1.6	1.8	2.3	2.0	2.1	1.7	1.9	1.4	1.4	1.3	1.8	1.8	1.5
	Worst	2.6	4.6	4.1	3.9	5.5	3.8	4.3	6.7	7.5	6.8	5.6	3.9	3.2	4.4	8.0	6.5	6.8	5.3
ONA	Mean	1.8	2.0	1.5	1.5	1.7	1.7	1.4	2.4	1.8	1.7	1.9	2.2	1.4	1.6	1.1	1.7	1.7	1.8
	Worst	6.2	5.2	6.3	4.4	3.8	5.9	3.6	6.3	5.9	5.8	5.8	6.9	4.5	4.0	6.8	6.2	4.9	5.3
OMM	Mean	1.2	2.0	2.2	2.0	1.8	2.4	2.4	2.2	1.7	1.8	1.9	1.7	2.2	2.0	1.2	1.6	1.9	1.8
	Worst	7.3	6.9	5.1	4.9	4.7	5.0	7.2	6.8	6.5	6.3	5.0	4.0	5.6	6.1	4.5	6.1	6.2	6.0
UT	Mean	1.9	1.4	2.0	1.6	1.6	1.7	3.1	1.7	2.4	2.2	2.6	1.7	2.6	2.3	1.6	1.8	2.1	1.9
	Worst	4.5	5.3	5.5	5.5	6.5	5.4	7.1	6.0	6.3	4.5	7.4	6.3	5.6	5.0	6.0	3.9	4.9	3.9
Hang	Mean	1.9	1.0	0.8	0.8	1.6	0.7	1.5	1.8	1.6	1.6	1.8	2.2	1.6	1.8	1.5	1.6	1.4	1.4
	Worst	5.4	2.8	2.5	2.4	5.4	2.3	3.2	4.8	4.5	4.1	6.9	4.5	5.0	4.7	4.9	4.3	4.8	4.1

improvement is negligible compared to the previous version. In short, compilers with better results regarding *Vanish* are LLVM-based ones. Although restricted to only two ISAs, these findings suggest a pattern for the adoption of LLVM-based compilers but additional experiments considering other ISAs would be necessary to explore further the reliability benefits of this compiler.

Lastly, we believe that the industry will not adopt a compiler set just because it is more reliable, except for a niche such as critical-safety applications (e.g. autonomous vehicle and medical applications). In this sense, we propose to evaluate which compiler set is more reliable and that also offers the best performance. Figure 5 shows a trade-off between performance (i.e. execution time from RTL and executed instructions from OVPsim) and reliability (i.e. the occurrence of *Vanishes*), considering the aforementioned cross-compilers. The upper-left corner presents the best of both performance and reliability; conversely, the lower-right corner has the worst performance and reliability. In turn, the lower-left corner assemblies are the configurations that show reasonable performance, but low soft error reliability.

Results show that the applications compiled with the *O0* flag presented a higher susceptibility to soft errors and a low performance, except for the codes generated with the *Arm 5.06* compiler, which showed a reliability improvement, but still, below to other sets of compilers. Although the *Arm 5.06* compiler provides GCC compatibility to aid development with source bases that were originally configured to be built with the GNU toolchains [35], the *Arm 5.06* performs a series of optimizations (e.g. reduction of memory access—*mem2reg*) even at the *O0* level. In some cases, such optimizations can reduce the number of static instructions to less than half w.r.t. to other compilers [36]. At the other end, are the *Arm 6.10* and *Clang 6.0.1* compilers (i.e. both LLVM-based). The two have the best set of performance and reliability. Among them, the *O2* optimization flag stands out, presenting superior results for the two compilers. However, due to the significance of our results (high confidence level and low margin of error), *Clang* has a certain advantage, and it would be our compiler suggestion. Finally, the remaining compiler sets maintain similar results to the mean of this trade-off.

4.4 | Closing Remarks

In this section, we demonstrated that the use of virtual platforms brings speedups greater than 1000 times for FI campaigns. Then, we saw that the architectural difference between the two Arm processors affects the system's reliability, producing different fault class behaviours. However, the inclusion of an operating system did not affect the soft error assessment accuracy. Next, we assessed the reliability of a variety of off-the-shelf compilers. We concluded that the best compilers are LLVM-based and that the best set in terms of performance and reliability would be the *Clang 6.0.1* compiler using the *02* optimization flag.

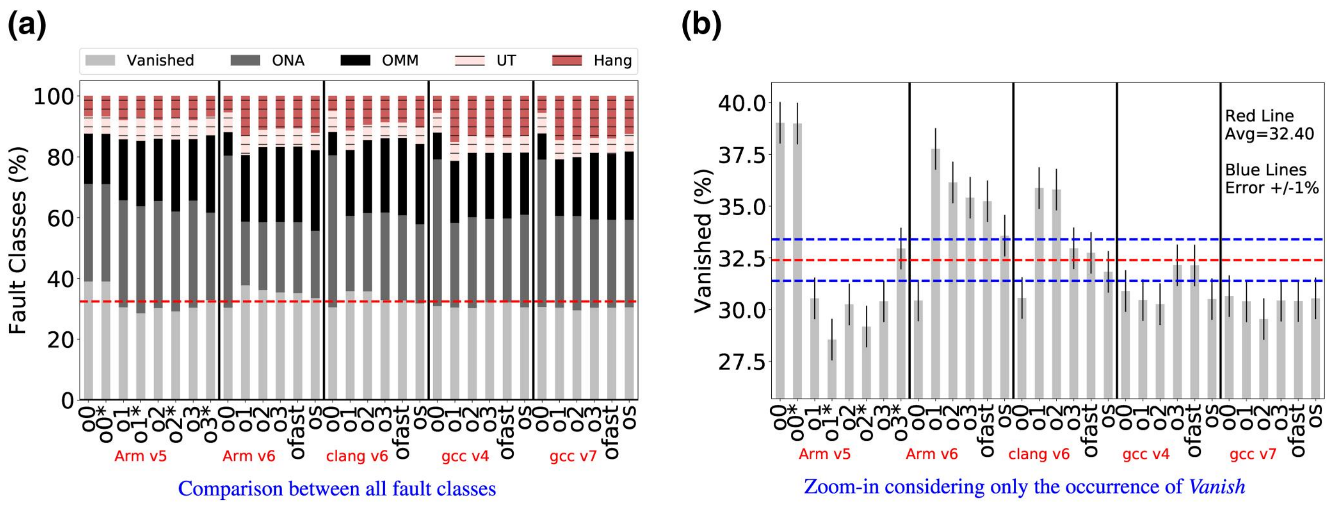


FIGURE 4 Average faults for compilers and their optimization flags. The ***** means a combined flag equivalent to *Ofast*

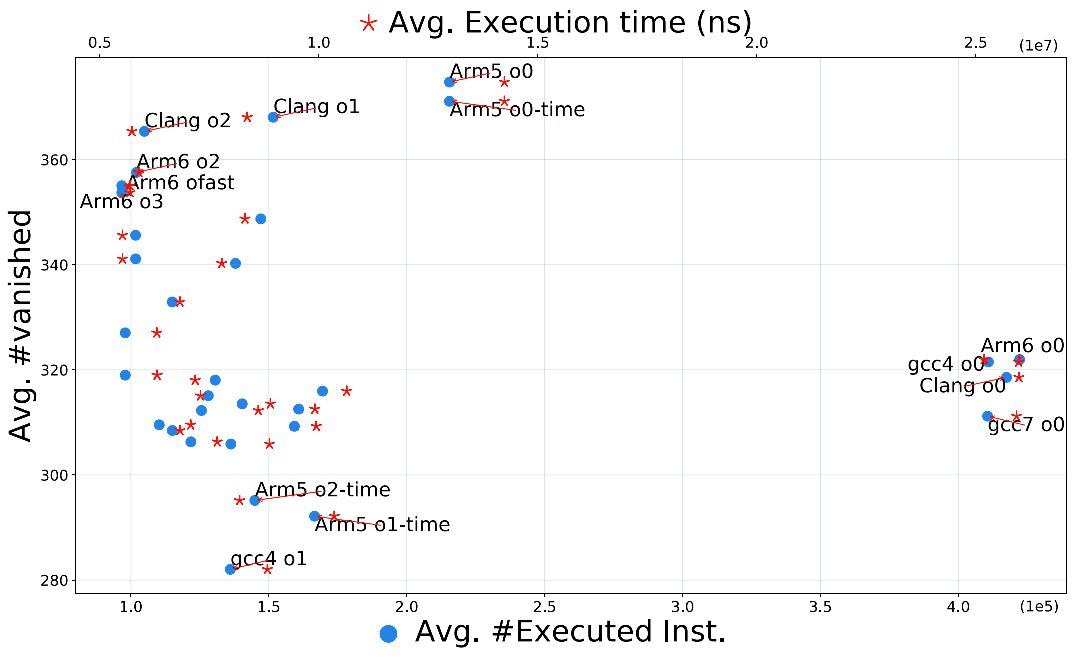


FIGURE 5 Average execution time and executed instructions versus number of Vanishes considering different compiler sets

5 | SOFT ERROR CONSISTENCY ASSESSMENT FOR MULTICORE PROCESSORS

Given the growing complexity of both application and software stacks, most computing systems consider multicore processors. Differently from single-core, in multicore processor architectures, the number of cores and the parallel programing model have a direct impact in terms of performance, power efficiency, and reliability. According to the previous Section, the software stack complexity, as well as the instruction set, slightly affects the accuracy of the soft error evaluation of OVPsim-FIM w.r.t. a RTL approach. Unfortunately, RTL descriptions of multicore processors are not freely available, and for that reason, this section considers the gem5 simulator [12] as the reference model.

In this scenario, this section evaluates the soft error consistency assessment of OVPsim-FIM using gem5-FIM as the reference model. Section 5.1 details the adopted experimental setup. Next, Section 5.2 presents the simulation speedup improvement achieved by OVPsim-FIM, detailing some virtual platform parameters. Then, Section 5.3 presents the assessment of soft error consistency considering three main aspects: number of cores (Section 5.3.1), different parallel programing models (Section 5.3.2), and different VP parameters that directly impacts on the soft error resilience (Section 5.3.3).

5.1 | Experimental setup

Table 3 presents the proposed experimental setup used to measure the soft error analysis consistency between the two VPs. The target processor is an Arm Cortex-A9 processor (Armv7 Architecture) because it can be configured to use one, two or four cores.

The proposed experimental setup adopts two distinct workloads: the Rodinia benchmark suite [26] and the NAS Parallel Benchmarks (NPB) [25]. The two benchmarks provide a set of applications that use different parallel programing models (i.e. OpenMP, MPI, CUDA, and OpenCL) designed to assess the performance of parallel supercomputers. This experimental setup considers 16 OpenMP Rondinia applications: (a) backprop, (b) BFS (Breadth-First Search), (c) heartwall, (d) hotspot, (e) hotspot3d, (f) kmeans, (g) lavaMD, (h) lud, (i) myocyte, (j) nn (k-Nearest Neighbours), (k) nw (Needleman-Wunsch), (L) particlefilter, (M) pathfinder, (N) sradv1, (O) sradv2, and (P) streamcluster. In addition, we also selected 11 NPB applications (from Serial, MPI, and OpenMP programing models): (BT) Block Tri-diagonal solver, (CG) Conjugate Gradient, (DC) Data Cube, (DT) Data Trafic, (EP) Embarrassingly Parallel, (FT) Discrete 3D fast Fourier Transform, (IS) Integer Sort, (LU) Lower-Upper Gauss-Seidel solver, (MG) Multi-Grid on a sequence of meshes, (SP) Scalar Penta-diagonal solver, and (UA) Unstructured Adaptive mesh.

To avoid external influences and ensure the most solid comparison between virtual platforms, the software stack of both Benchmarks uses the same compilation environment regarding compiler (*GCC 6.2.0*), optimization flag (*-O3*), libraries, and target an identical Linux kernel (*version 3.13.0-rc2*). Operating system reliability is not the main focus of this section and, therefore, fault injections only occur during the application lifespan (i.e. the OS startup is not subject to faults). Nevertheless, the operating system calls that arise during this period (i.e. application execution time) are susceptible to fault injections as part of the application's behaviour.

TABLE 3 gem5 versus OVPsim Experimental Setup

Processors	**Arm Cortex-A9 with 1, 2, and 4 Cores**
Benchmarks	Rodinia [26] and NPB [25]
Programing Models	Serial, MPI, and OpenMP
Number of Applications	27
Injections per Scenario	8000
OVP Quantum Sizes	448,000, 4,480, 448, and 44
Total Fault Injections	1,248,000 (Section 5.3.1) and
	1,072,000 (Section 5.3.2) and
	1,888,000 (Section 5.3.3)

5.2 | FI simulation performance of OVPsim-FIM w.r.t. gem5-FIM

Simulation speedup is one of the main reasons for adopting VPs, however, their engines make them different from each other, providing more or less accuracy according to their performance. For example, gem5 is an event-based cycle-accurate simulator, that is it describes the target micro-architecture as components (register-file, pipeline, cache, etc.) interconnected by a series of events. A scheduler in the gem5 engine executes these events at each simulation tick, updating the whole system state. One tick corresponds to 1 picosecond; therefore, for a 2 GHz CPU clock, events are executed at a rate of 500 ticks per CPU cycle in the simulated system and, consequently, a complete instruction requires a few thousand ticks. The resulting clock cycle accuracy is obtained at the expense of a higher computation and memory cost.

Due to the high simulation speed (typically at hundreds of MIPS), virtual platform simulators based on just in time (JIT) dynamic binary translation, such as OVPsim, appear to have an advantage over event-driven simulators since it translates the target ISA (e.g. Armv7) to host x86-64 instructions. Further, a complete instruction is the OVPsim minimal simulation granularity; in other words, the simulation always advances one instruction, which provides a higher simulation speed than gem5. Similar to an OS scheduler in which several processes share the same CPU time, the OVPsim engine simulates each model instance (i.e. processor, core, peripheral) for a fixed-length instruction block called *Quantum*.

The quantum size is a measure of relative time for a give processing unit (e.g. core, CPU) given in the number of executed mnemonic instructions (e.g. add, load, store). Consequently, one quantum of 100 requires the execution of one hundred instruction fetches from the memory (i.e. compiled object code).The compiled object code consists of assembly-level mnemonic execution, thus, both represent the same magnitude. The engine always completes the entire instruction per iteration, so if one complex assembly-level mnemonic is decomposed into multiple instructions during the decoding phase (internally to the core state machine), for accountability purposes, the instructions counts as one. The quantum size is configurable using a variable, *time-slice*, representing a time in seconds that refers to an internal configuration parameter and not to the simulation, host, or real-time. The quantum size is given by Equation (3), where, by default, the time-slice is 0.001 s (1 ms) and the target processor's nominal MIPS rate (proc. MIPS) is 448 MIPS, resulting in a quantum size of 448,000 instructions, being our reference size in Table 3.

$$QuantumSize = (proc.\ MIPS) \times 1e^{6} \times (time\ slice\ duration) \tag{3}$$

The OVPsim also deploys a scheduling policy to manage the simulation of processors and other components. For example, the simulator selects the first processor, after it has been simulated for 448,000 instructions, it is suspended, and the next processor assumes. In the case of multicore processors, such as Arm Cortex-A9x2, each processor core receives a separate quantum and the simulation is scheduled accordingly. Figure 6 shows two simulation scenarios for a dual-core processor, one with the default quantum, and the other using half of its size (i.e. 224,000 instructions). By reducing the quantum size, the number of model switches for an identical workload increases. This configuration choice delays inter-core communication (or synchronization events) and consequently reduces their simulation speedup.

To quantify the simulation speedup difference between gem5-FIM and OVPsim-FIM, we analysed multiple configurations and workloads on a Quad-core Intel Core I7-7700K 4.2 GHz with 16 GB DDR4 2400 MHz. Figure 7 presents the results from one to four cores, showing the scalability and speedup of supported parallel simulations by the two VPs.

Figure 7 a displays the first set of simulations considering the Rodinia applications. In this experiment, using four cores, the gem5-FIM atomic simulation speedup ranges from 4.2 to 11 MIPS (Figure 7a), while the detailed model reaches from 0.89 to 1.65 MIPS. In turn, the OVPsim-FIM ranges from 345 to 2921 MIPS depending on the quantum size and application. Note that the reduction in quantum size decreases the number of instructions executed per block, directly affecting the simulator performance due to the increasing switching between the models (i.e. cores).

In addition, Figure 7 b presents NPB applications, which are larger than Rodinia's, reaching up to 87 billion instructions. Figure 7 b shows a better performance of OVPsim-FIM in all configurations while the gem5-FIM atomic remains stable. Looking at the scenario with four cores, the longest workload reaches 12.5 MIPS using atomic gem5. On the other hand, OVPsim-FIM reaches 3910 MIPS, approximately 312 times faster. The OVPsim-FIM simulation speedup increases as the application grows due to the just-in-time engine algorithm, thus benefiting from larger applications. For example, comparing the larger and smaller applications, the simulation speedup ranges from 1190 to 3910 MIPS (i.e. an increase of 3.28 times) where the gem5 atomic difference is less than 20%, ranging from 10 to 12 MIPS.

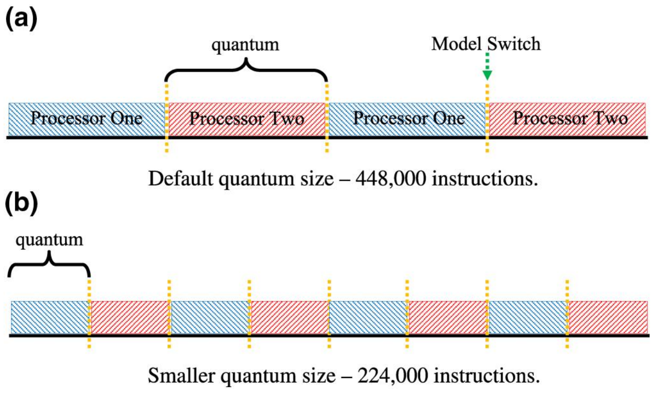

(a)
quantum
Model Switch
Processor One
Processor Two
Processor One
Processor Two
Default quantum size – 448,000 instructions.
(b)
quantum
Smaller quantum size – 224,000 instructions.

FIGURE 6 OVPsim scheduling policy varying the quantum size for a dual-core processor executing the same workload

5.3 | Soft error mismatch analysis

This section presents the assessment of soft error resilience in multicore processors, comparing OVPsim-FIM and gem5-FIM. First, we assess the impact due to the number of cores (Section 5.3.1). Then, we investigate which programing model would be the most reliable (Section 5.3.2). Finally, in Section 5.3.3, we discuss how OVPsim-FIM configuration (i.e. quantum size) affects the soft error resilience.

5.3.1 | Mismatch analysis considering the number of cores

In the last decade, multicore architectures have been gaining prominence in several semiconductor sectors, being found today in automobile, medical and consumer electronic devices. Due to its importance, engineers must understand how architectural choice affects the system's reliability. With this intention, we analysed the Arm Cortex-A9 for the Rodinia and NPB benchmark suites. For simplicity, this analysis considers only OpenMP-based applications; the difference between the programing models is analysed in Section 5.3.2. Figure 8 presents a detailed multicore mismatch between the OVPsim-FIM (OVP) and the gem5-FIM atomic (GA), considering Rodinia (Figure 8a–c) and NPB applications (Figure 8d–f). Concerning the two benchmarks, Rodinia applications execute on average 80 million instructions, while NPB applications execute on average 17 billion instructions (i.e. 212x larger). The longer NPB execution reduces the probability of *ONA* due to a higher likelihood of a bit masking when compared to Rodinia applications. This behaviour is seen in Rodinia applications that reach more than 5% of *ONA* mismatch (F, I, L, and P). Furthermore, Rodinia applications have a higher number of *Hangs* than NPB applications, increasing notably with the number of cores. The two possible causes are (*i*) the fault affected a loop statement (e.g. while, for): where a longer execution translates to more significant recovery time; and (*ii*) kernel malfunctions: the fault injection leads to unrecoverable kernel perturbations (e.g. a thread scheduler error). A longer execution time reduces the Linux kernel exposure time (i.e. the probability of kernel function be stroke by a fault). In other words, the longer the applications, proportionally, the fewer kernel functions are executed. In short, longer workloads reduce overall mismatch. NPB applications' average mismatch varies from 1.32% to 1.68%, in contrast to Rodinia, which ranges from 1.39% and 2.63%. Furthermore, the worst-case mismatch between the gem5-FIM and OVPsim-FIM is reduced to up to 55% for NPB applications.

Analysing the presence of multicore architectures, applications show a worsening mismatch while increasing the number of cores, most notably in the Rodinia applications B, D, and K (Figure 8) and in the NPB applications MG and CG (Figure 8). In the worst-case, Rodinia's mismatch increases to 17.71% using quad-core compared to 6.06% using single-core processors. In addition, the average error grows from 1.39% to

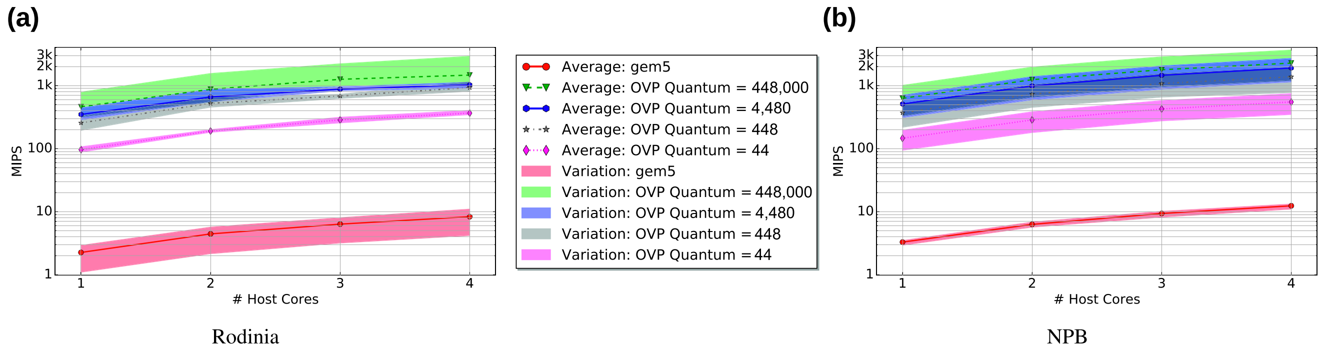


FIGURE 7 Simulation speedup and scalability of the two virtual platforms, showing the performance gain achieved by using the OVPsim

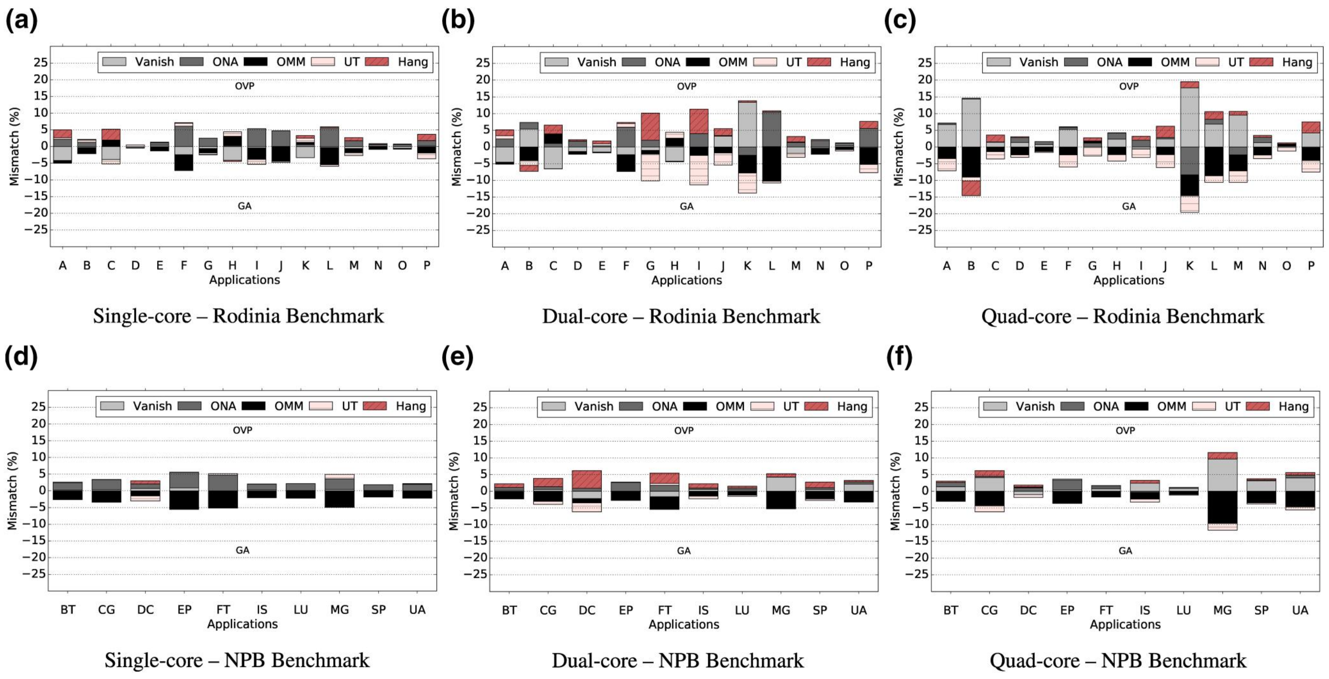


FIGURE 8 Mismatch between gem5-FIM and OVPsim-FIM varying the number of cores in Arm Cortex-A9

2.51% considering one and two cores and remains stable for four cores with a mismatch of 2.63%. Increasing the core count results in more thread context switching. For Rodinia applications, this behaviour combined with sub-linear scalability (i.e. underutilised cores) leads to further errors in the kernel. This happens because, during CPU downtime, the OS executes the scheduler algorithm by running one application at a time (i.e. no other threads are running), and then moves to a sleep mode (i.e. *waiting for the interruption*). Furthermore, the number of *Vanish* increases in multicore architectures (2 and four cores). This difference can be attributed to the intercore communications. Due to its instruction-accurate engine, the OVPsim-FIM simulation time (i.e. the number of instructions executed) is affected by the running application characteristics (Section 5.3.3 details this behaviour by modifying parameters of the OVPsim-FIM simulator).

5.3.2 | Mismatch analysis considering parallel programing Models

The emerging use of multicore processors requires specialised libraries that include additional complexity in the soft error assessment. Regarding this distinction, this section considers NPB applications to assess the mismatch between our reference *Serial* programing model on a single-core processor with the *OpenMP* and *MPI* programing models based on multicore architectures. The OpenMP library uses a series of fork and join approaches to parallel loop statements, in which the API automatically creates children threads, being suitable for shared memory. On the other hand, the MPI standard is adequate for distributing systems due to the use of a message-oriented parallelisation technique, which requires direct parallelisation of the user in relation to the creation and communication of threads.

The introduction of parallel programing models increases the software stack, which makes some components more critical to the correct behaviour of the system. For example, injecting faults into a thread scheduling function has a potentially more hazardous effect on system reliability than a purely arithmetic code portion. By comparing the active periods of these critical functions with the application's execution time, it is possible to define a time interval called the *vulnerability window*, which varies with the number of calls and executions of the function. In this sense, the use of the NBP benchmark suite provides a real high-performance workload, allowing a more accurate assessment of the impact of the OpenMP and MPI libraries on the system's reliability. Due to its reduced vulnerability window, the parallelisation mechanism has a limited effect on the final reliability assessment, less than 23% in the worst-case. To assess this reliability, Figure 9 shows FI campaigns and mismatches comparing MPI and OpenMP applications.

First, we compare the *Serial* implementation with the two parallelisation libraries, considering a single-core processor. The purpose is to assess how each software stack affects the susceptibility to soft errors. For each application, we assess 8000 FI campaigns, which means that our results have a confidence level of 95% and a margin of error of 1.1%. Comparing with both parallel programing models, no significant variation was found for single-core execution, that is the fault distribution is within the margin of error in most applications. However, some applications follow the same pattern, while BT, CG, and IS have fewer soft errors in the *Serial* implementation; EP, FT, and SP have fewer soft errors when increasing the software stack using either of the two parallel programing models.

On the other hand, when we compare the two parallel programing models in a multicore system, we see that out of 27 possible scenarios between the MPI (Figure 9b) and OpenMP (Figure 9c), in 22 the MPI has a higher masking rate (i.e. executions without any errors). This is due to two main reasons: *First*, MPI applications have a better workload balance among the used cores, that is, the number of executed instructions per core is very similar. For instance, the average difference concerning executed instructions per core is around 4% considering MPI applications, while the OpenMP variation reaches up 16%. As the OpenMP does not fully utilise the available cores due to the fork/join parallelisation approach where a loop statement executes in parallel and other code portions hastily, corroborating the results presented in [14]. By contrast, the MPI has individual and independent working threads for each running core providing a better workload balance during its execution. Whenever a core is sub-utilised, it executes a thread scheduling policy, and when no thread is suitable, the core waits in a sleep mode. By consequence, the kernel relative exposure time and its probability to suffer a transient fault increases, as the scheduling is more often executed. *Second*, OpenMP benchmarks have a shorter execution time, 16% on average, compared against the MPI applications. By consequence, diminishing the vulnerability window of the MPI inner-functions when comparing against the OpenMP. Further, the longer execution increases the chance of the injected fault being erased due to the software and microarchitectural masking mechanisms. These results show that MPI should be prioritised over OpenMP to improve the reliability of multicore systems.

5.3.3 | Mismatch analysis considering the OVPsim-FIM quantum parameter

This section explores the impact of using distinct quantum-sized configurations (i.e. 448,000, 4,480, 448, and 44 instructions per block) in the OVPsim-FIM to assess the soft errors' reliability of singe and multicore processors. This analysis provides the trade-off between the performance discussed in Section 5.2 with the reliability accuracy brought by each configuration, making it easier for engineers to understand and choose the best configuration for their purposes.

Figure 10 shows the reference FI framework, the gem5-FIM atomic (ψ), compared to OVPsim-FIM using four quantum sizes 448,000 (λ), 4480 (γ), 448 (β), and 44 (δ) instructions per block for single-, dual-, and quad-cores in an Arm Cortex-A9, respectively. Figure 10 shows that the variation in quantum size affects the accuracy of the soft error resilience results of OVPsim-FIM. As expected, the best performing configuration (i.e. λ—first column of each application shown in Figure 10) is the one with the greatest mismatch with the results provided by the reference (i.e. ψ—last column of

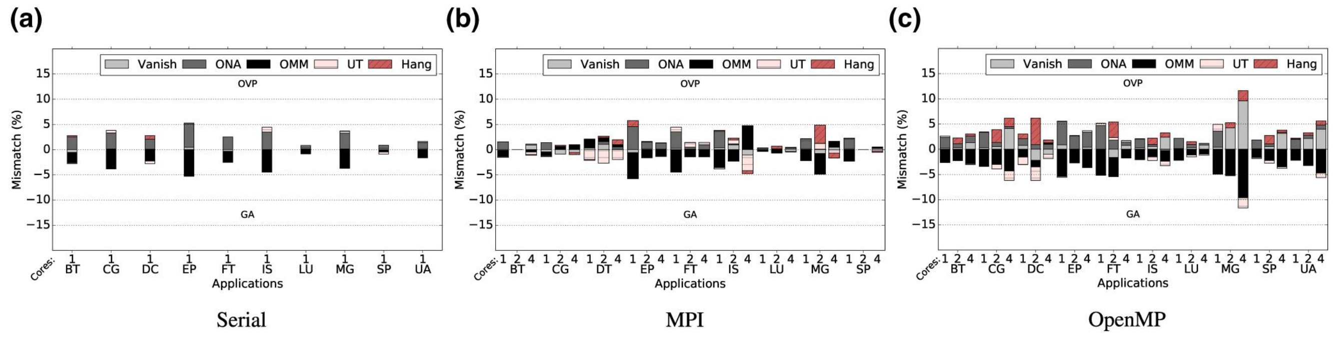


FIGURE 9 Mismatch between gem5-FIM and OVPsim-FIM considering different programing models

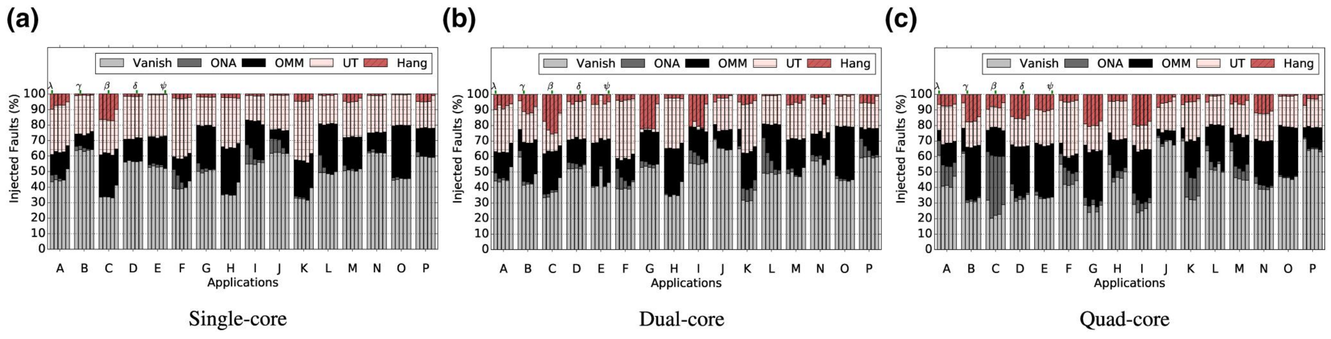


FIGURE 10 Mismatch between the gem5-FIM (*ψ*) and OVPsim-FIM quantum sizes: 448,000 (*λ*); 4480 (*γ*); 448 (*β*); and 44 (*δ*)

each application shown in Figure 10). On the other hand, the one with the lowest performance of OVPsim (i.e. *δ*), which is at least 31× faster than the reference in quad-core simulations according to Section 5.2, presents the closest values. For example, the quad-core processor model (Figure 10c) has an average improvement of 40% in resilience accuracy when using the smallest block (*δ*) instead of the largest block (*λ*). This is due to some applications that have a large mismatch with the reference results (e.g. B and K). Note that the reduction of the quantum size decreases the communication cycles between cores, approaching the OVPsim-FIM and gem5-FIM behaviours. Another behaviour is the migration from *ONA* to *OMM* by decreasing the quantum size, in other words, the incorrect content previously restricted to the register file migrates to the end of the memory.

The resulting mismatch shown in Figure 10 can be traced back to its block-based simulation engine, as discussed in Section 5.2, where each core executes a fixed amount of instructions before moving to the next one. Note that inter-core communications are completed during the core switch, leading to temporally unsynchronised cores. Inter-core communication is necessary to synchronize events across multiple cores, for instance, in a parallelisation library. Rodinia OpenMP-based applications use a fork-join parallelisation paradigm, where synchronization barriers coordinate multiple children threads execution. One synchronization event that requires all cores to reach the same statement (i.e. a barrier) requires multiple quantum executions to complete. Delaying these communication events lead to some extra instructions executed by OVPsim-FIM due to other cores waiting. So, we investigate how programing models behave with quantum size changes, as shown in Table 4.

Table 4 shows the mismatch between gem5-FIM and OVPsim-FIM using the default quantum size (DF) of 448,000 instructions and a smaller quantum size (Q) of 44 instructions, considering four distinct workloads: NPB Serial, NPB MPI, and NPB OpenMP, along with Rodinia OpenMP applications. The comparison of the two quantum sizes (i.e. DF vs. Q) is highlighted, where *green* means that the mismatch reduces with the decrease of the quantum size; *black* when both are equivalent, and *red* when the mismatch increases with the decrease of the quantum size.

The results in Table 4 show that in the vast majority of cases, the reduction in the quantum size decreases the mismatch between gem5-FIM and OVPsim-FIM. For the Rodinia suite, this behaviour is kept, and it is further accentuated in the quad-core system due to the instruction count reduction, as previously mentioned. For example, the average mismatch reduced from 2.63% to 1.08%, as shown in the last row of Table 4. In the same scenario, the worst-case mismatch result decreased from 17.71% to only 3.64%. Interestingly, we have one case where the reduction of the quantum size causes a worsening in the accuracy of the soft error resilience, which is for NPB MPI in a quad-core system. In this scenario, the results derived from Table 4 show an average difference of 0.17% (i.e. 0.80–0.69%) and a worst-case of 0.9% (i.e. 5.59–4.69%).

In order to understand where the mismatches shown in Table 4 come from, Figure 11 shows the accuracy difference presented by Rodinia (Figure 11a) and NPB applications (Figure 11b) when OVPsim-FIM is configured with the smallest quantum size (i.e. a 44-instruction block). Regarding the migration from *ONA* to *OMM*, this trend is very clear when looking at Rodinia's applications A, F, I, L, and P, compared to Figure 8 that is configured with the default quantum size. In addition, the FI campaigns simulated with the OVPsim-FIM(Q) presents a mismatch reduction in 24 out of 26 scenarios (except Rodinia's applications C and O) with a significant (5×) improvement in the worst-case of OpenMP-based benchmarks, which is justified by the impact of synchronization barriers between children threads. The most significant reductions are for Rodinia's application K, using a quad-core system, reducing from 20% to 4%; and MG application of the NPB benchmark also in a quad-core system, reducing from 12% to less than 2.5%.

Regarding the benchmarks' difference, Rodinia includes applications with up to 220 *million*, while NPB applications vary from 16 to 87 *billion* instructions. By consequence, NPB benchmarks have more extended computations between synchronization points than the Rodinia, which impacts on the soft error assessment accuracy. NPB benchmark also has a better workload distribution and scalability, which means in conjunction with the more prolonged execution that children threads have enough instructions to complete one or more quantum blocks between OpenMP barriers. On the other hand, Rodinia applications have less computation between synchronization points, sometimes shorten then one quantum execution, leading OpenMP barriers to executing extra instructions while waiting for other threads. The Rodinia behaviour magnifies the mismatch originated due to the OVPsim-FIM simulation policy using fixed-length instructions blocks and, as a consequence, these applications benefit more by reducing the quantum size, achieving an accuracy gain up to 5×, as seen in Figure 11.

5.4 | Closing Remarks

This section presents a complete multicore soft error resilience assessment. Simulation performance is fundamental for early design space explorations, for this purpose, this section started by showing that the peak OVPsim-FIM simulation speed is around 4000 MIPS, considering a quad-core system. This proves that JIT-based FI frameworks are efficient means to assess the soft error resilience early in the design phase. Next, results demonstrate that applications showed a worsened mismatch while increasing the number of cores. However, this mismatch can be mitigated by using MPI, which has been shown to bring the best results in terms of reliability accuracy. Finally, we show that OVPsim-FIM Quantum parameter affect the soft errors reliability assessment and that a good trade-off between simulation performance and reliability accuracy would be using a small quantum size, such as a 44-instruction block.

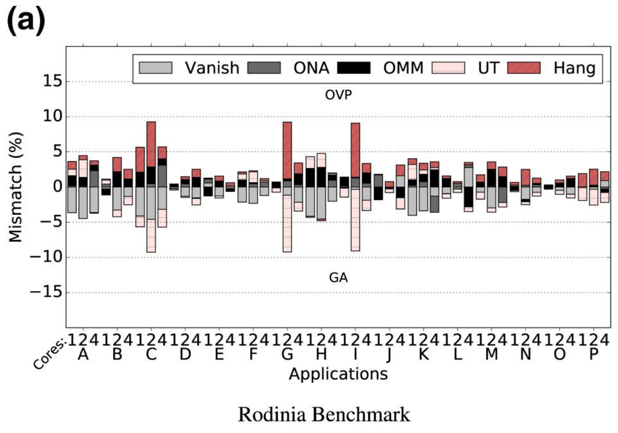
TABLE 4 Mismatch comparison of programing models of OVPsim-FIM with default (DF) and small quantum size (Q) in relation to gem5-FIM

#	Workload and Programing Models	Single-core (%) DF	Single-core (%) Q	Dual-core (%) DF	Dual-core (%) Q	Quad-core (%) DF	Quad-core (%) Q
Worst case	NPB Serial	5.24	3.51	*	*	*	*
	NPB MPI	5.17	1.19	4.08	1.64	4.69	5.59
	NPB OpenMP	5.39	2.06	5.27	1.67	9.61	2.25
	Rodinia OpenMP	6.06	4.16	13.35	8.76	17.71	3.64
Best case	NPB Serial	0.01	0.01	*	*	*	*
	NPB MPI	0.01	0.00	0.01	0.00	0.01	0.01
	NPB OpenMP	0.01	0.01	0.01	0.00	0.01	0.01
	Rodinia OpenMP	0.03	0.00	0.04	0.01	0.01	0.01
Average	NPB Serial	1.15	0.55	*	*	*	*
	NPB MPI	1.06	0.26	0.83	0.35	0.63	0.80
	NPB OpenMP	1.32	0.52	1.42	0.41	1.68	0.42
	Rodinia OpenMP	1.39	0.82	2.51	1.53	2.63	1.08

6 | CONCLUSIONS

This work has investigated the soft error assessment consistency of a JIT virtual platform simulator (OVPsim-FIM) with more than 12 million fault injections considering single and multicore Arm processor architectures. The fault injection campaigns considered different cross-compilers, software stacks, programing models, and 52 applications. Results demonstrated that the architectural difference, such as the ISA, between the two Arm processors, affects the reliability of single-core systems. However, the addition of tiny operating systems (e.g. FreeRTOS) in software stack did not affect resilience accuracy. Regarding cross-compilers, those based on LLVM appeared to be more reliable ones, with the best compiler set being the *Clang 6.0.1* using the *02* optimization flag.

Furthermore, we showed a worsened mismatch while increasing the number of cores, that can be mitigated a little by using the MPI programing model. Finally, we demonstrated that by tuning the OVPsim-FIM for a more detailed simulation by the quantum size parameter (i.e. 44-instruction block), we obtained the best cost-benefit in terms of soft error assessment accuracy, with a worst-case mismatch of 8.76% and high simulation performance, reaching up to 345 MIPS. Authors



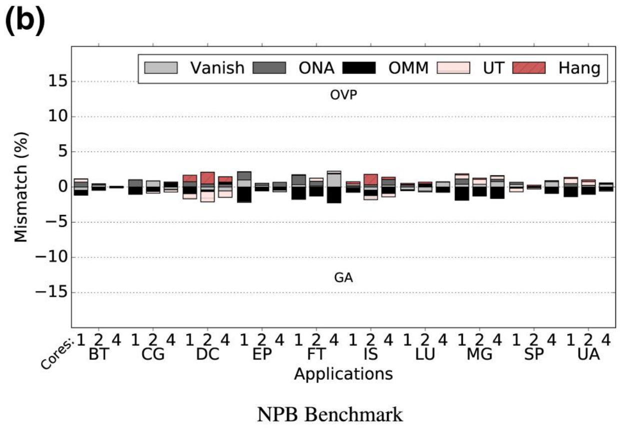


FIGURE 11 Multicore mismatch between gem5-FIM and OVPsim-FIM with smallest quantum size (a 44-instruction block)

conclude that achieved mismatch are acceptable and are not a hindrance to evaluate soft errors at early design phases. Furthermore, given the remarkably achieved speedup, the utilization of JIT-based FIM appears promising since it can also be used for comparison among different processor models, ISAs, kernel and complex benchmarks with billion instructions. Finally, authors also believe that the high statistical significance presented gives to this work the potential to be a reference for other studies with concerns about soft error resilience of Arm processors.

Future works include further investigations of the soft error consistency of JIT-based FI frameworks considering more realistic scenarios such as FPGA-based fault injection approaches, which emulates the occurrence of faults by modifying the bitstream configuration. Authors also intend to compare some of the failure-related data sets reported in this article with results obtained from neutron radiation tests.

ORCID

Geancarlo Abich https://orcid.org/0000-0001-9387-1523
Rafael Garibotti https://orcid.org/0000-0002-7307-0128
Vitor Bandeira https://orcid.org/0000-0001-7459-0072
Felipe da Rosa https://orcid.org/0000-0003-4964-5136
Jonas Gava https://orcid.org/0000-0001-7113-6448
Felipe Bortolon https://orcid.org/0000-0001-6288-978X
Guilherme Medeiros https://orcid.org/0000-0001-9842-1644
Fernando G. Moraes https://orcid.org/0000-0001-6126-6847
Ricardo Reis https://orcid.org/0000-0001-5781-5858
Luciano Ost https://orcid.org/0000-0002-5160-5232

REFERENCES

1. Baumann, R.: Soft errors in advanced computer systems. IEEE Des. Test Comput. 22(3), 258–266 (2005)
2. Li, T. et al.: Processor design for soft errors: challenges and state of the art. ACM Comput. Surv. 49(3) (2016)
3. Times, E.: Toyota case: single bit flip that killeds. http://www.eetimes.com/document.asp?doc_id=1319903 (2015). Accessed November 2020
4. Cho, H. et al.: Quantitative evaluation of soft error injection techniques for robust system design. In: Design Automation Conference (DAC). Austin, TX, USA, June, pp. 1–10 (2013)
5. Rosa, F. et al.: Evaluation of multicore systems soft error reliability using virtual platforms. In: International New Circuits and Systems Conference (NEWCAS). Strasbourg, France, June, pp. 85–88 (2017)
6. Parasyris, K. et al.: A fault injection tool for studying the behavior of applications on unreliable substrates. In: International Conference on Dependable Systems and Networks (DSN). Atlanta, GA, USA, June, pp. 622–629 (2014)
7. Rosa, F. et al.: A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability. In: International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS). Amherst, MA, USA, October, pp. 211–214 (2015)
8. Kaliorakis, M. et al.: Differential fault injection on microarchitectural simulators. In: International Symposium on Workload Characterisation (IISWC). Atlanta, GA, USA, October, pp. 172–182 (2015)
9. Chatzidimitriou, A. et al.: Demystifying soft error assessment strategies on ARM CPUs: microarchitectural fault injection vs. neutron beam experiments. In: International Conference on Dependable Systems and Networks (DSN). Portland, OR, USA, June, pp. 26–38 (2019)
10. Mansour, W., Velazco, R.: SEU fault-injection in VHDL-based processors: a case study. J. Electron. Test. 29(1), 87–94 (2013)
11. Abbasitabar, H., Zarandi, H.R., Salamat, R.: Susceptibility analysis of LEON3 embedded processor against multiple event transients and upsets. In: International Conference on Computational Science and Engineering (CSE), Paphos, Cyprus, December, pp. 548–553 (2012)
12. Binkert, N., et al.: The gem5 simulator. SIGARCH Comput. Archit. News. 39(2), 1–7 (2011)
13. Imperas: Open virtual platforms (OVP). http://www.ovpworld.org/ (2020). Accessed October 2020
14. da Rosa, F. et al.: Extensive evaluation of programming models and ISAs impact on multicore soft error reliability. In: Design Automation Conference (DAC). San Francisco, CA, USA, June, pp. 1–6 (2018)
15. Lins, F.M. et al.: Register file criticality and compiler optimization effects on embedded microprocessor reliability. IEEE Trans. Nucl. Sci. 64(8), 2179–2187 (2017)
16. Sangchoolie, B. et al.: A study of the impact of bit-flip errors on programs compiled with different optimization levels. In: European Dependable Computing Conference (EDCC). Newcastle upon Tyne, UK, May, pp. 146–157 (2014)
17. Medeiros, G. et al.: Evaluation of compiler optimization flags effects on soft error resiliency. In: Symposium on Integrated Circuits and Systems Design (SBCCI). Bento Gonçalves, RS, Brazil, August, pp. 1–6 (2018)
18. Hoste, K., Eeckhout, L: COLE: compiler optimization level exploration. In: International Symposium on Code Generation and Optimization (CGO). Boston, MA, USA, April, pp. 165–174 (2008)
19. Serrano Cases., A et al.: Nonintrusive automatic compiler-guided reliability improvement of embedded applications under proton irradiation. IEEE Trans. Nucl. Sci. 66(7), 1500–1509 (2019)
20. Wei, J. et al.: Quantifying the accuracy of high-level fault injection techniques for hardware faults. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, Atlanta, GA, USA pp. 375–382 (2014)
21. Lu, Q. et al.: Llfi: an intermediate code-level fault injection tool for hardware faults. In: 2015 IEEE International Conference on Software Quality, Reliability and Security, IEEE, Vancouver, BC, Canada pp. 11–16 (2015)
22. Schirmeier, H., Breddemann, M.: Quantitative cross-layer evaluation of transient-fault injection techniques for algorithm comparison. In: European Dependable Computing Conference (EDCC), Naples, Italy, September, pp. 15–22 (2019)
23. Schirmeier, H., Borchert, C., Spinczyk, O.: Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, RJ, Brazil, September, pp. 319–330 (2015)
24. Patel, A. et al.: 'MARSS: a full system simulator for multicore x86 CPUs'. In: Design Automation Conference (DAC), San Diego, CA, USA, June, pp. 1050–1055 (2011)
25. Bailey, D.H., et al.: The NAS parallel benchmarks summary and preliminary results. In: ACM/IEEE Conference on Supercomputing, Albuquerque, NM, USA, November, pp. 158–165 (1991)
26. Che, S., et al.: Rodinia: a benchmark suite for heterogeneous computing. In: International Symposium on Workload Characterization (IISWC). Austin, TX, USA, October. pp. 44–54 (2009)
27. Gustafsson, J. et al.: The mälardalen WCET benchmarks: past, present and future. In: International Workshop on Worst-Case Execution Time Analysis (WCET). Brussels, Belgium, July, pp. 136–146 (2010)
28. Bortolon, F.T. et al.: Exploring the impact of soft errors on noc-based multiprocessor systems. In: International Symposium on Circuits and Systems (ISCAS), Florence, Italy, May, pp. 1–5 (2018)
29. Mukherjee, S.S. et al.: A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: International Symposium on Microarchitecture (MICRO), San Diego, CA, USA, December, pp. 29–40 (2003)

30. Seifert, N., et al.: Soft error susceptibilities of 22 nm tri-gate devices. IEEE Trans. Nucl. Sci.. 59(6), 2666–2673 (2012)
31. Leveugle, R. et al.: Statistical fault injection: quantified error and confidence. In: Design, Automation and Test in Europe Conference (DATE), Nice, France, April, pp. 502–506 (2009)
32. ARM: Arm University Programme (2020). http://www.arm.com/support/university. Accessed July 2020
33. Machado, R.S. et al.: Comparing performance of C compilers optimizations on different multicore architectures. In: International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Campinas, SP, Brazil, October, pp. 25–30 (2017)
34. Song, L. et al.: COMP: compiler optimizations for manycore processors. In: International Symposium on Microarchitecture, Cambridge, UK, December, pp. 659–671 (2014)
35. Limited A. ARM Compiler gnu_version Flag. https://developer.arm.com/documentation/dui0472/m/compiler-command-line-options/--gnu-version-version?lang=en (2016). Accessed July 2020
36. Limited AArm compiler getting started guide. https://developer.arm.com/documentation/dui0529/latest/ (2016). Accessed July 2020

How to cite this article: Abich G, Garibotti R, Bandeira V, et al. Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator. *IET Comput. Digit. Tech.* 2021;15:125–142. https://doi.org/10.1049/cdt2.12017