Accurate FIT Rate Estimation Through High-Level Software Fault Injection

Pablo R. Bodmann Daniel Oliveira Paolo Rech

Abstract—**Reliability is today one of the major issues for computing devices from the embedded domain up to large High-Performance Systems. To safely deploy a computing device in a system, accurate measurements of the FIT rate are required to ensure that the device complies with the project reliability requirements. In this work, we present a method to provide a more accurate device FIT rate estimation from software fault injection. We use the technology sensitivity factor, obtained from beam experiments, as well as architectural and code features to significantly increase the FIT rate estimation accuracy. We compare the estimated FIT rates with the ones measured from radiation experiments of eight codes executed on ARM Cortex-A5 and Cortex-A9. We show that, on average, we can provide a FIT rate estimation accuracy of less than 20% (overestimation) and 35% (underestimation) the expected FIT rate for A5 and A9, respectively.**

Index Terms—**reliability, soft errors, failures in time, neutron beam, fault injection, FIT rate estimation**

I. Introduction

Reliability is one of the most critical concerns from consumer applications to the automotive, military, aerospace, and High-Performance Computing (HPC) markets. Several fields in which computation is critical, such as self-driving cars, airplanes, or Unmanned Aerial Vehicles (UAVs), require high reliability and unexpected errors should be strictly avoided [1]. Additionally, reliability has been listed as one of the top ten challenges to reach exaFLOPS scale [2] as transient errors lead to lower scientific productivity and significant monetary loss [3].

The reliability of computing devices has been extensively studied [4]–[7] and the Failure In Time (FIT) rate has been accurately measured for several applications on modern devices. While one can dramatically reduce the FIT rate of components with radiation hardening techniques [8], such methods require expensive radiation-hardened components with performance and applicability limitations. On the other hand, several computing products are available in the consumer market offering efficiency and delivering high performance at a very low cost. For automotive and HPC applications, for instance, besides reliability, performances and costs are also main constraints. Thus, Commercial-Off-The-Shelf (COTS) computing solutions are very attractive for these applications. The use of COTS devices is acceptable for high-reliability applications as long as their FIT rate is accurately evaluated. In cases where the FIT rate exceeds the reliability requirement, some software-level or system-level hardening solutions can still be applied to reduce the device error rate.

The three main strategies to evaluate hardware and software reliability are field tests, radiation experiments, and high-level software fault injection. Field testing is a very expensive and time-consuming approach since a large number of devices must be employed. Radiation experiments use accelerated particles beam and are also expensive. However, beam testing provides a realistic FIT rate requiring few devices and a relatively short testing time. High-level software fault injection, in contrast, is much more versatile and its cost can be negligible. The two key advantages of software fault injection are speed and a detailed code sensitivity analysis. The major drawback of high-level software fault injection is the limitation to access some hardware resources not available at the software level. Then, fault injection cannot easily mimic natural phenomena, like radiation experiments, making the FIT rate estimation a challenging task.

Our main contribution with this paper is a methodology to improve the accuracy of the FIT rate estimation obtained using high-level software fault injection. Besides the software fault injection results, we use the device's technology sensitivity factor, which is the device's raw sensitivity to neutrons. We use the Level 1 cache cross-section as representative for the technology sensitivity factor. It is worth noting that, while this factor needs to be measured with radiation experiments, as it is a property of the device, the same value can be used for all programs of interest for a specific device. The software fault injection provides the probability of a fault propagating to the output manifesting as an error. Then, with a combination of architectural and code features, extracted with profiling tools, we can estimate a much more accurate FIT rate for a program executed on a given device. We evaluated eight codes executed on two ARM devices, Cortex-A5 and Cortex-A9. Our FIT rate estimation, on average, yields an accuracy of 20% (overestimation) and 35% (underestimation) from the FIT rate measured using beam experiments on the A5 and A9 respectively.

The rest of the paper is organized as follows. Section II summarizes the background and reports the related work. Section III presents a detailed description of the adopted

This paper was submitted for review on the 1st of October 2021. This research has been supported in part by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 886202 and from the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001. Neutron beam time was provided by ChipIR (DOI: 10.5286/ISIS.E.RB2000161) thanks to C. Cazzaniga, M. Kastriotou, and C. Frost.

Pablo R. Bodmann is a Ph.d. student at the Informatics Institute on the Federal university of Rio Grande do Sul, Porto Alegre, Brazil (e-mail: prjbodmann@inf.ufrgs.br).

Daniel Oliveira is a professor at the Department of Informatics on the Federal University of Parana, Curitiba, Brazil (e-mail: @inf.ufrgs.br)

Paolo Rech is an assistant professor at the Industrial Engineering Department of the University of Trento, Italy (email: paolo.rech@unitn.it) and an associate professor at the Federal university of Rio Grande do Sul, Porto Alegre, Brazil (email: prech@inf.ufrgs.br)

methodologies. In Section IV we discuss and compare the results obtained with beam and fault injection, evaluating the accuracy of FIT rate estimation. Finally, Section V draws conclusions.

II. Background and Related Work

In this section, we present background material and related works on processors' reliability and their assessment methodologies.

A. Radiation Effects in Electronic Devices

The Earth is constantly bombarded by high-energy particles coming from space. These particles interact with Earth's atmosphere and that interaction produces a flux of several different particles, being mainly neutrons. About 13 $neutrons/((cm^2) \times hour)$ reach the ground [9]. When a neutron hits a transistor, the strike may perturb its state which in turn generates bit-flips in memory, or produces current spikes in logic circuits that, if latched, lead to an error [10]. A transient error may not affect the program output (i.e., the fault is masked, or the corrupted data is not used) or may be propagated through the stack of system layers leading to a Silent Data Corruption (SDC - output is corrupted without any indication), or Detected Unrecoverable Errors (DUEs), such as a program crash (application hang) or a device turning not responding or rebooting (system crash). The probability of an error to occur in a code being executed on a microprocessor depends on the memory/logic sensitivity [11], the probabilities for the fault to be propagated through the hardware design (the microarchitecture), and the program [12].

B. Reliability Evaluation Methodologies

There are several different ways to evaluate the reliability of computing devices. The most commonly used are live or field tests, beam experiments, and fault injection at different levels of abstractions (software, microarchitecture, RTL). Table I lists the main characteristics of these evaluation methodologies, including the time and cost required to complete the study, how many of the available resources can be accessed, if the faults are realistic or synthetic (i.e., models chosen by the user), if the study can be performed in the early stages of the project or only on the final product, and how detailed is the performed evaluation.

Field tests are done by exposing devices employed in the field to the natural particles' flux. The FIT rate can be obtained using either a dedicated test-bench, as Xilinx's Rosetta project [13], or data coming from user devices or production machines, as supercomputers [14]. While field tests can provide the most accurate error rate, they are extremely expensive because a huge number of devices must be employed to have sufficient data and are very time-consuming, since the natural error rate is very low.

Accelerated particles beams can help to reduce the cost and time of tests by taking advantage of a much higher particles flux intensity, while still mimicking the neutron energy spectra [11], [15]. While the measured error rate is very realistic, unfortunately, beam experiments have two limitations: (1) faults can be observed only when they have already compromised the system functionality, making it very challenging to identify the most vulnerable parts of the system. (2) Experiments can be performed only in special facilities that house particle accelerators.

Fault-injection is another approach to evaluate the reliability of a device. Faults are injected by the user at different levels of abstractions: from Register-Transfer Level (RTL) [16] to microarchitecture [17] and software [18]. By injecting faults in the accessible resources of each level's model (gates, registers, hardware arrays, variables, instructions, etc.) it is possible to measure the probability for a fault to propagate at the output of an application. Fault-injection has two main limitations: (1) the fault model and fault injection probabilities are synthetic (i.e., defined/modeled by the user and/or the simulator), thus the obtained results risk being unrealistic, and (2) faults can be injected only in that subset of available resources that are accessible. Typically, RTL fault injection can, in principle, access any gate or storage element but it is, unfortunately, prohibitively slow to allow large programs reliability analysis; Microarchitecture was demonstrated to be very accurate when compared to beam [4]. However, this type of injection can only be made with a microarchitectural description of the target CPU. Software fault injection is fast but can access only a very limited set of resources (the ones accessible by the user).

To overcome the software fault injection limitations, we include software and hardware features such as variable lifetime and cache sizes. We show that we can significantly improve the error rate estimation by systematically combining the features insights with fault injection results. A developer could use these easily obtainable metrics and tools to have an early estimation of the FIT rate. A timely and sufficiently accurate FIT rate estimation can then be helpful in tuning the system under development. Given the technology sensitivity of the hardware, the strategy we propose can replace costly and time-consuming beam testing for a preliminary investigation of the system error rate. This, potentially, can diminish the test cost and time-to-market.

C. Related Work

ARM Cortex-A9 processors have been exposed to accelerated particles beam and have been subject to fault injection in previous studies [19]. [20] and [21] present results on architecture-level fault injection of the processor core, while [22] includes a microarchitecture-level fault injection on A9. [23] presents a comparative reliability evaluation between microarchitecture and RTL fault injection, for bare-metal workloads running on Cortex-A9, while [17] also includes results of RTL fault injection on ARM CPU cores. Also, [4] compares both microarchitecture fault injection and beam experiments and shows that FIT estimation using microarchitecture is very close to the beam.

Some work has been done to evaluate the influence of an OS on the reliability of code executions [19]. It is shown that the OS can be beneficial in the presence of cache conflicts. However, these papers do not perform fault injection, as we do

TABLE I
RELIABILITY EVALUATION METHODOLOGIES CHARACTERISTICS.

Evaluation Method	Time Needed	Cost	Accessible Resources	Fault Model	Availability	Observability
Field/Life	months/years	very high	all	realistic	final product	limited
Beam	**hours**	**high**	**all**	**realistic**	**final product**	**limited**
SW Fault-Injection	**hours**	**low**	**limited**	**synthetic**	**final product**	**medium**
RTL Fault-Injection	months	low	all	synthetic	early	very high

in this paper, but are just based on beam experiments, which might limit the insights that can be gathered.

Finally, in this work, we combine and compare the reliability evaluation performed using beam experiments and with statistical fault injection on top of software fault injection. This gives the programmer a powerful tool for estimating the final FIT rate.

III. METHODOLOGY

In this section, we describe the devices under test, the codes we use, the radiation experiment, and fault-injection setups. Finally, we present the methodology to estimate the error rate from the software fault injection results.

A. Codes and Devices Under Test

Both beam experiment and fault injection campaigns are performed on an ARM®Cortex™-A5 implemented in a 65nm CMOS technology in the Microchip SAMA5D2 XPLAINED ULTRA board and on the ARM®Cortex™-A9 that is embedded in a Xilinx Zynq™-7000 SoC implemented in a 28nm CMOS technology. The two ARM CPUs have significantly different microarchitectures: the A5 is an in-order CPU while the A9 is an out-of-order. The silicon chips also differ: the Microchip features a stand-alone A5 while the Xilinx features an A9 integrated on an SoC. The SAMA5D2 device has a single-core operating at a maximum frequency of 500MHz and the Zynq SoC has two ARM A9 cores operating at 667 MHz and an FPGA (not used in our tests). Each core has 32 KB instruction and data caches and a unified Level 2 cache which is 128kB in the A5 and 512kB in the A9.

We have chosen codes with different computational characteristics from the *mibench* [24] benchmarks suite. The chosen codes are CRC32, FFT, Matrix Multiplication (128x128 and 400x400), Qsort, and Susan C, E, and S (used in image processing). For each code, we use the very same input (size and value) for both beam experiment and fault injection.

B. Neutron Beam Experiments

Radiation experiments were performed at the ChipIR facility of the Rutherford Appleton Laboratory (RAL) in Didcot, UK. ChipIR delivers a neutron beam suitable to mimic the atmospheric neutron effects in electronic devices [25], allowing to measure the Failures In Time (FIT) rate of the device executing a code.

Figure 1 shows part of our setup at ChipIR. We irradiate two Xilinx Zedboards and three Microchip boards with a 3×3 cm beam spot, which is sufficient to irradiate the chip uniformly without affecting the main memory or other onboard peripherals (data in the DDR is not exposed to the beam).

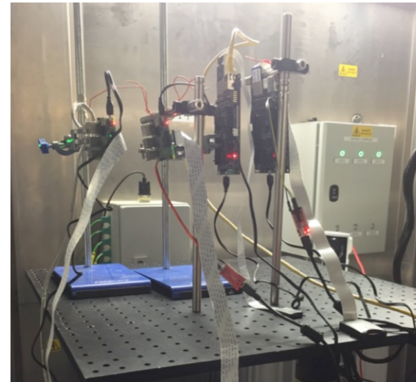

Fig. 1. Part of the beam test setup at ChipIR.

The available neutron flux was about $3.5 \times 10^6 n/(cm^2/s)$.

We have carefully designed the experiments not to have more than a single corruption during a program execution (observed error rates were lower than 1 error per 1,000 executions).

Each of the 16 configurations (8 codes per device) was tested for at least 100 effective hours (i.e., not considering setup, initialization, and recovery from crash times). The 1,600 hours of the test, when scaled to the natural exposure, account for more than 10 million years.

C. Software Fault Injection

We chose CAROL-FI [15] to inject faults at the software level, measuring the Program Vulnerability Factor (PVF) [26]. PVF is the probability of an injected fault generating an SDC or Crash). We inject faults using the CAROL_FI random fault model, overwriting every bit of a variable by a random bit. As the fault is generated at a high level, we are considering transient faults that, by propagating from the hardware level, change the value of a memory location.

This choice is dictated by the fact that we intend to simulate also the effect of faults in the instructions that update the variables in which the fault is injected, not just errors in the memory location storing the variable (for which a single/double bit flip suffice). This methodology has already been demonstrated to be accurate in simulating transient faults in parallel applications [15].

CAROL-FI allows also to isolate the PVF contribution of each variable. We use these features to refine the error rate prediction, as shown in the next subsection.

D. Cross Section Estimation

The error rate of a code executed on a device depends on several factors. The principal ones are (1) the probability for

a fault to occur in the hardware (the technology cross-section) and (2) the amount of resources actually used for computation, and (3) the probability for the fault to affect the software and reach the output (the PVF).

We have experimentally measured the per bit cross-section of the L1 cache on the A5 and the A9 to be $1.82 \times 10^{-14} cm^2$ and $1.99 \times 10^{-15} cm^2$, respectively. We use these values as representative of the technology sensitivity of the two devices. We choose the L1 cache as it is one of the most critical resources in ARM devices [4]. It is worth noting that caches can be protected with error correction or detection codes. Then, deriving the factor from protected caches may underestimate the fault rate of unprotected memory cells (e.g., register banks and buffers inside the chip). On the other hand, caches may have the highest bit cross-section, leading to overestimation for the remainder chip components.

Software fault injection provides the probability for a fault in a variable to affect the computation. However, to predict the error rate of the code, we still need to estimate the probability for the variable memory space to be alive and exposed to neutrons. We use the following performance and architectural metrics to adjust the fault injection prediction:

- **Size**
  The PVF of each injected variable is multiplied by its size to consider the memory area (a larger area implies a higher probability for the variable to be corrupted).
- **Lifetime**
  The bluepercentage of time that the variable is alive during the program execution. We adjust the PVF for each variable by multiplying it by the variable's lifetime.
- **Cache size**
  We consider data in the main memory to be fault-free since most DDR chips include ECC. In our experiment, we carefully leave DDR out of the beam spot. Thus, only data in the caches can be corrupted. If the data required by a code exceeds the cache size, then some variables must be evicted and stored in the DDR, where they are protected. To take this effect into account, we introduce the f_{cache} factor. We consider the total size of the m variables that are alive at the same time during code execution (this is a snapshot of the data currently being used). If this value is larger than the size of the L2 cache, we know that some variables are evicted. We then divide the L2 cache size by the sum of the variable's total size, resulting in a normalization factor that considers the probability for a variable to be in the cache, that we call f_{cache}. This factor is lower than one since a variable may be evicted during its lifetime. If all variables fits in the cache, then $f_{cache} = 1$.

$$f_{cache_i} = \begin{cases} (L2_{size} / \sum_{j=1}^{m} size_j), & \text{if } \sum_{j=1}^{m} size_j > L2_{size} \\ 1, & \text{otherwise} \end{cases}$$

- **IPC**
  Superscalar devices may use additional hardware to improve performances (i.e., more than one operation is being executed). We use the Instructions Per Cycle (IPC) metric to take into account the computational efficiency (how many variables are being processed). Higher IPC means that more variables are being updated.

We considered four error rate estimations applying incrementally each factor described above. As a first evaluation, **Estimation 1**, we estimate the cross section of a code (σ_{code}) that has n variables by multiplying the cache per bit cross section (σ_{tech}) with the number of bit of variable i ($size_i$) and its PVF_i (measured with fault injection):

$$\sigma_{code} = \sum_{i=1}^{n} \sigma_{tech} \times PVF_i \times size_i \tag{1}$$

Then, to have a more accurate estimation, **Estimation 2**, we also consider the life time of variables (LT_i):

$$\sigma_{code} = \sum_{i=1}^{n} \sigma_{tech} \times PVF_i \times size_i \times LT_i \tag{2}$$

To take variables eviction from caches into account, we introduce the cache factor (f_{cache_i}). The third cross section estimation, **Estimation 3**, then becomes:

$$\sigma_{code} = \sum_{i=1}^{n} \sigma_{tech} \times PVF_i \times size_i \times f_{cache_i} \times LT_i \tag{3}$$

Finally, to consider the computational efficiency we multiply the estimation from Eq. 3 by the code's Instructions Per Cycle, **Estimation 4**:

$$\sigma_{code} = IPC \times Equation\ 3 \tag{4}$$

It is worth noting that all metrics are easily measured by profiling or debugging the program, which requires a negligible time to compute. For this work, we use *GDB*, *Gprof*, and *perf* tool. The cache size is obtained from the device's specification. Using the estimated cross-section and the real neutron flux of NYC, we calculate the FIT rate.

IV. Results and Discussion

In this section, we first present the software fault injection and beam experiment results. Then, we compare the code sensitivity ranking for the beam test and software fault injection (i.e., ordering the codes from the most sensitive to the least one). A similar ranking across fault injection methodologies indicates that a relative comparison between codes is feasible, identifying which codes will perform better. We also quantify how the FIT rate obtained from software fault injection is similar to the one from beam experiments. We show that some architectural and code features, such as cache size and variables lifetime, greatly improve the accuracy of FIT rate estimation.

A. High-Level Fault Injection and Beam Experiments Results

Figure 2 and Figure 3 show the PVF measured with high-level fault injection and the FIT rate measured with Beam experiments respectively.

The PVF is the probability for the fault to propagate to the output. Thus, the PVF assumes that the fault has occurred and

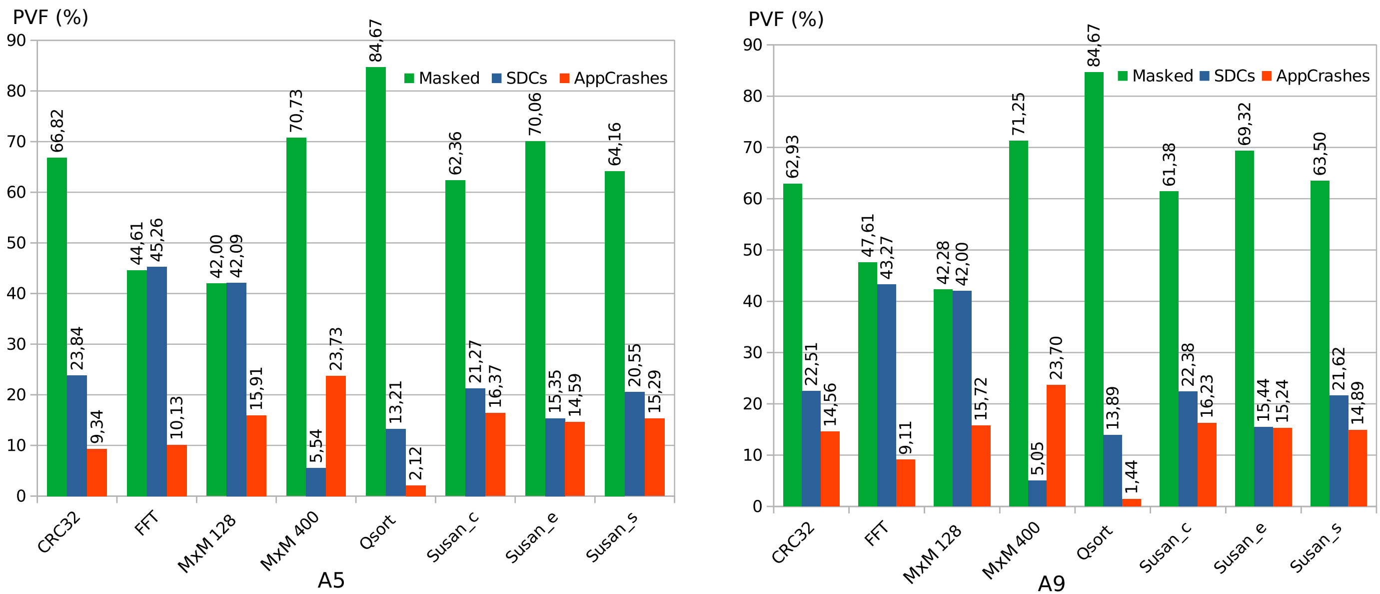

Fig. 2. Program Vulnerability Factor measured with high-level fault injection. The PVF assumes that the fault has occurred and tracks its propagation to the output. A higher PVF indicates that it is more likely for the fault to propagate to the output.

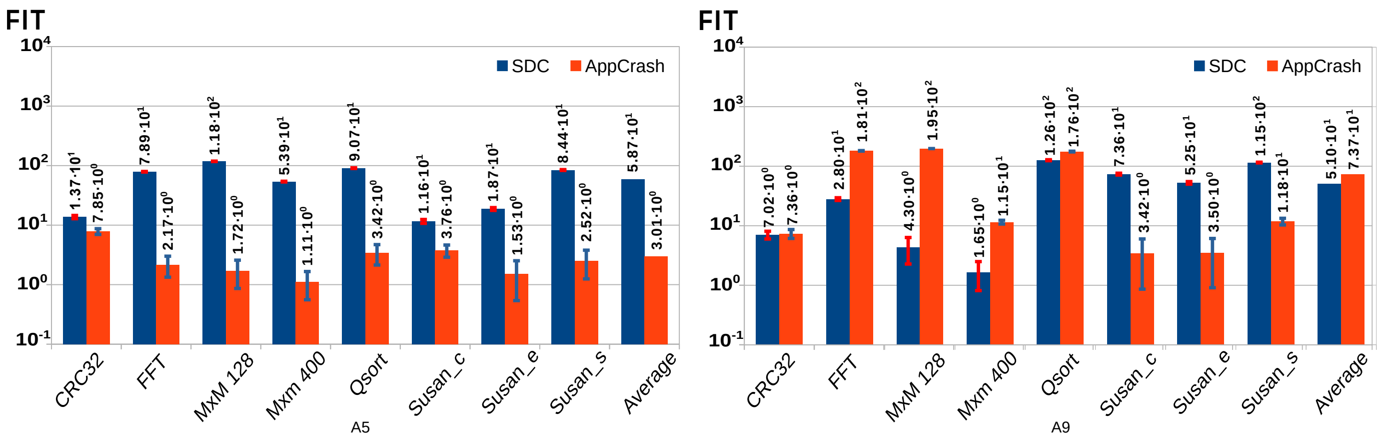

Fig. 3. FIT rates values measured with beam experiments. The FIT rates encloses all reliability factors, from the technology sensitivity to the amount of resources used for computation and the probability for the fault to propagate (i.e., the PVF).

indicates how likely it is to affect the computation. The PVF gives no indication about neither the raw device sensitivity nor the amount of resources used for computation. Looking at Figure 2, it can be seen that most of the faults injected are masked, i. e., do not result in either a SDC or a crash. This can be explained by the fact that the variables are either overwritten after the injection, are not going to be used for reaching the result, or are logically masked downstream (e.g., multiplied by zero). Also, it is interesting to note that, for most benchmarks, the crash PVF is smaller than the SDC PVF. Additionally, there is a significant difference between the PVF of the various codes, but the PVF for a single code on both devices is practically identical. Fault injection evaluates mostly the code, which is generated using the same algorithm and compiler for both devices with small variations due to the micro-architecture. Thus, it is expected that similar codes with the same input data will produce similar PVF results.

The FIT rate, shown in Figure 3, is the number of errors expected in 10^9 hours of operation. Given the beam energy spectrum characteristics, we can estimate the error of the device in a realistic application rate by multiplying the experimental cross-section with the neutron flux at NYC (i.e., 13 neutrons per cm^2 per hour) [9]. SDCs are detected by comparing the code output of the irradiated device with the expected golden output obtained executing that code before irradiation. A mismatch is marked as SDC. An Application Crash occurs when the code finishes with an error or hangs for longer than double the normal execution time.

Since the whole chip is irradiated, the FIT rate considers all the variables that affect the sensitivity of the device executing a code, including the code PVF. Nevertheless, as errors are observed only at the output, it is not possible, with beam experiments alone, to identify the variable that has the most impact on the code FIT rate. When analyzing the result for the beam experiments, it can be observed that *the average SDC rates are very similar for the two devices* (the average SDC

rate difference between the A5 and A9 is 13%). Interestingly, *the average Appcrash rate is at least one order of magnitude higher on the A9*. This is expected since the A9 is bigger and more complex (out-of-order and with a four times larger L2 cache) than the A5, so it is likely to have a higher probability of being hit by a neutron. Moreover, in contrast to the fault injection PVF, the most sensitive code for one device is not the same for another one. The FIT rate takes into account the algorithm as well as more features than fault injection, such as components inaccessible to software and device utilization. Thus, the FIT rate of the same algorithm and input data can significantly change among devices.

B. SDC Code Sensitivity Ranking

To have a first evaluation of the difference between beam experiments and software fault injection we present the ranking of codes, based on their SDC sensitivity, measured with beam experiments and predicted through software fault injection (using the estimation methods with variable size, lifetime, cache utilization and IPC, described in Section III-D). We want to understand if software fault injection is able to identify the code with the highest error rate.

Table II and III rank the codes from the most sensitive to the least one with respect to the SDC FIT rate. This gives us a comparison of sensitivity for each code. It is expected that a code should be as sensitive in the beam as on the fault injection. The first column presents the ranking from the beam experiments, which is the ranking expected in a real environment. The four next columns depict the ranking considering the software fault injection's FIT rate multiplied incrementally by the factors described in section III-D.

The sensitivity ranking for beam experiments shows Qsort as a highly sensitive code for both devices, which is expected since bit-flip masking is less probable for an in-place sorting algorithm. On the other hand, CRC32 is among the least sensitive for both devices. CRC32 uses fewer resources in both devices, processing a large input sequentially. Furthermore, MxM 128 is more sensitive than MxM 400, indicating that a larger input decreases the probability for a bit-flip to propagate to the output, due to the input being outside the caches which are more sensitive than the main memory. The Susan_s reports the highest sensitivity among the Susan family of codes in both devices. However, the least sensitive for the Susan codes changes for both devices with the corner detection as the least sensitive for A5, and the edge detector for the A9. Detecting corners and edges is very similar, thus, the complexity of each device and the machine code generated can be a decisive factor for the sensitivity ranking.

The software fault injection can identify Qsort as one of the most sensitive codes for both devices, regardless of the factors multiplying the FIT rate estimation. CRC32 is also identified as one of the least sensitive ones. However, the ranking for MxM and Susan versions are incorrect in both devices for the first factors multiplying the FIT rate estimation. For A5, shown in Table II, the size and lifetime factor are insufficient to explain the sensitivity behavior in beam experiments. However, including a third factor, cache size, is sufficient to obtain a very similar ranking. On the other hand, the IPC factor could not improve the estimation, indicating that the IPC factor may depend on the hardware in which the application is executed.

For the A9 device, in contrast to A5, software fault injection is unable to obtain a very similar ranking to the beam one, mostly due to the MxM codes. However, aside from MxM, the A9 obtain a similar ranking after including three factors (i.e., size, lifetime, and cache size), especially considering the most and least sensitive ones. The A9 is a more complex device capable of superscalar out-of-order execution. Then, a more fine analysis, to compensate for the limitations of software fault injection, is required to explain more precisely the behavior in the beam experiments. The IPC factor does not affect the ranking. However, as we will see in the next section, IPC improves the estimation for A9 as an adequate factor to take into account the extra sensitive area of superscalar cores.

C. High-Level Fault Injection and Beam Experiments Comparison

Figure 4 presents a comparison between the SDC FIT rate directly measured from beam experiments and the SDC FIT rate estimated from the software fault injection's PVF, as detailed in Section III-D.

The baron the right show the average FIT for all codes. It is worth note that we inject more then $2,000$ faults in each code, and the error bars are at most 5% for a 95% confidence level. The only exception is CRC32 on the A9 with a error bar interval of 12%.

The naive FIT rate estimation is computed by multiplying the PVF by the technology factor, then we show the FIT rate estimation multiplied incrementally by the four different factors detailed in Section III-D. The blue bars show the FIT rate estimation using the first factor only, the red bars use the first two factors, the yellow bars multiply by the first three-factor, and finally, the green bars use all four factors. The cyan bar is the FIT rate measured with beam experiments.

The first two methods for FIT rate estimation yield, on average, a poor accuracy for both devices. Including the cache size factor, the results improve significantly and reduce about two to three times the difference from the expected FIT rate. Thus, the cache size, limiting the number of variables in such a sensitive area, must be considered to achieve reasonable accuracy. It is worth noting that most of the errors occur in the caches, especially in the L2 [4].

Furthermore, for out-of-order superscalar cores such as the A9 cores, the IPC metric is necessary to consider how much of the extra hardware components are utilized for computation (i.e., instruction and thread-level parallelism). Thus, the IPC hints at the size of the sensitive chip area the code utilizes and may improve the FIT rate estimation accuracy. However, IPC is useful only if the code is compute-intensive and excites the device. As shown in Figure 4, for $Matmul$ the use of IPC improves the FIT prediction accuracy, but, on average, it actually deteriorates the A9 results. This means that a specific set of features should be carefully chosen for a certain class of algorithms and devices to obtain the most accurate results.

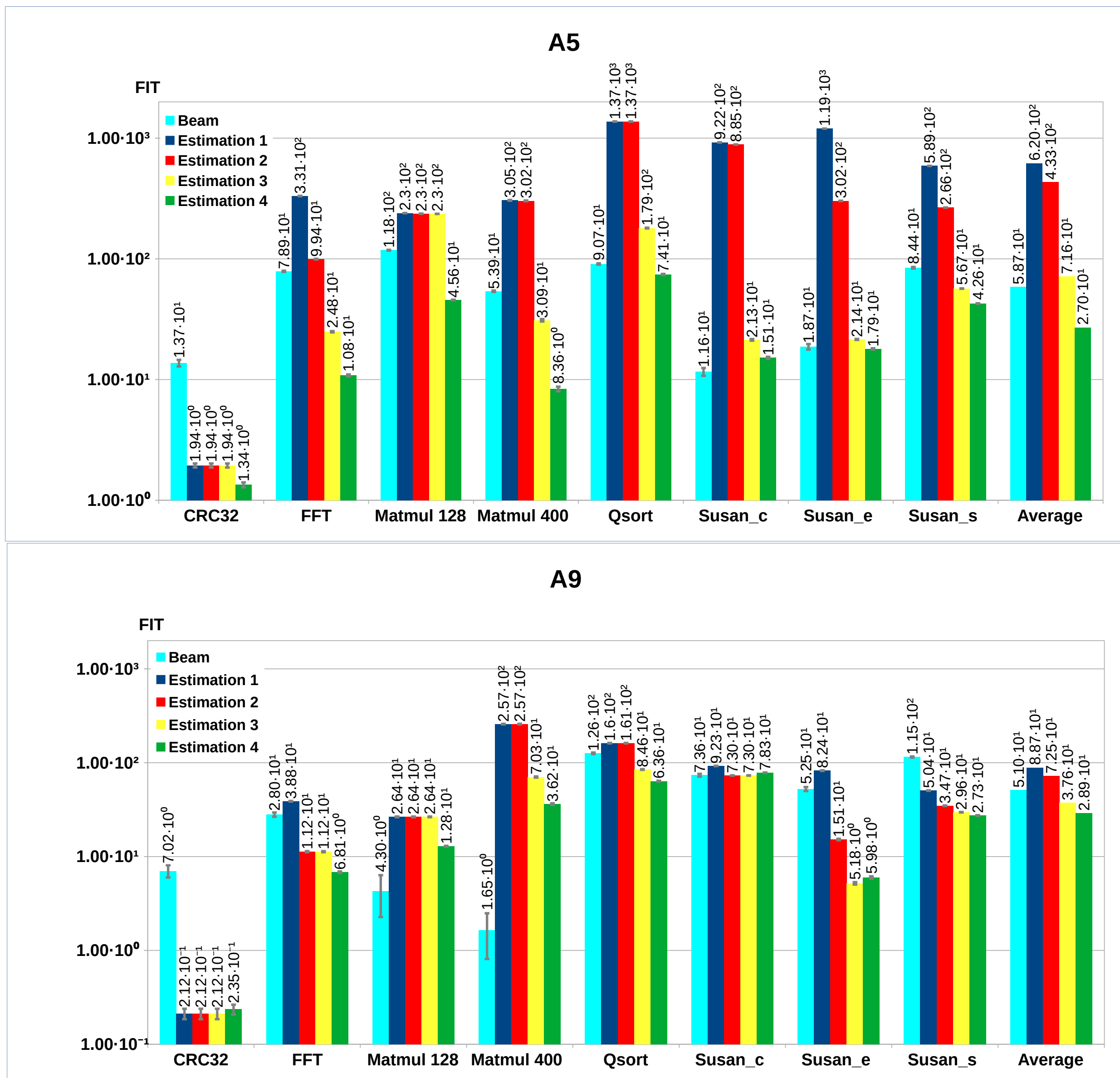

Fig. 4. Comparison of SDC FIT rate measured by beam experiments and estimated by software fault injection. The estimation methodology is detailed in Section III-D.

The CRC32 code shows a slight improvement of the FIT rate estimation accuracy on the A9 as we include factors, in contrast to other codes. CRC32 is a simple code reading a file char by char and is mostly unaffected by the four factors used in this work. Moreover, the file contents are stored in the operating system buffers. Then, due to the limitations of the software fault injection that is unable to access data outside of the executing program, we cannot inject faults in the file contents.

D. Discussion

Our analysis shows that we can improve the accuracy of the FIT rate estimation of an application using software fault injection with a combination of architectural and code features.

The first factor, the size of a variable, is not sufficient to improve significantly the FIT rate estimation. The software fault injection provides the probability of a fault propagating to the output of the application. However, the probability of corruption and data size, due to the hardware organization and complexity, is unable to explain how much data are susceptible to corruption (i.e., how much data are loaded to sensitive hardware components such as caches). Thus, features taken from the memory hierarchy and variable lifetime, for instance, are required to further improve the estimation accuracy.

Moreover, most of the features chosen in this work proved to significantly improve the accuracy of FIT rate estimation. The best estimation, on average, is estimation 3 for both devices, which is 20% (overestimation) accurate for A5 and 35% (underestimation) for A9. This result indicates that code and architectural features can indeed be used to better understand the reliability of software, improving the FIT rate estimation. Unfortunately, a generic model composed of the same set

TABLE II
A5 CODE SENSITIVITY RANKING, FROM THE MOST SENSITIVITY TO THE LEAST ONE. THE FIRST COLUMN IS THE EXPECTED RANKING MEASURED FROM BEAM TEST, AND THE FOLLOWING COLUMNS SHOW THE RANKING FROM THE FOUR DIFFERENT ESTIMATIONS DETAILED IN SECTION III-D

Expected (Beam Test)	FIT (Beam Test)	Estimation 1	Estimation 2	Estimation 3	Estimation 4
MxM 128	$1.18 \cdot 10^2$	Qsort	Qsort	MxM 128	Qsort
Qsort	$9.07 \cdot 10^1$	Susan_e	Susan_c	Qsort	MxM 128
Susan_s	$8.44 \cdot 10^1$	Susan_c	MxM 400	Susan_s	Susan_s
FFT	$7.89 \cdot 10^1$	Susan_s	Susan_e	MxM 400	Susan_e
MxM 400	$5.39 \cdot 10^1$	FFT	Susan_s	FFT	Susan_c
Susan_e	$1.87 \cdot 10^1$	MxM 400	MxM 128	Susan_e	FFT
CRC32	$1.37 \cdot 10^1$	MxM 128	FFT	Susan_c	MxM 400
Susan_c	$1.16 \cdot 10^1$	CRC32	CRC32	CRC32	CRC32

TABLE III
A9 CODE SENSITIVITY RANKING, FROM THE MOST SENSITIVITY TO THE LEAST ONE. THE FIRST COLUMN IS THE EXPECTED RANKING MEASURED FROM BEAM TEST, AND THE FOLLOWING COLUMNS SHOW THE RANKING FROM THE FOUR DIFFERENT ESTIMATIONS DETAILED IN SECTION III-D

Expected (Beam Test)	FIT (Beam Test)	Estimation 1	Estimation 2	Estimation 3	Estimation 4
Qsort	$1.26 \cdot 10^2$	MxM 400	MxM 400	Qsort	Susan_c
Susan_s	$1.15 \cdot 10^2$	Qsort	Qsort	Susan_c	Qsort
Susan_c	$7.36 \cdot 10^1$	Susan_c	Susan_c	MxM 400	MxM 400
Susan_e	$5.25 \cdot 10^1$	Susan_e	Susan_s	Susan_s	Susan_s
FFT	$2.80 \cdot 10^1$	Susan_s	MxM 128	MxM 128	MxM 128
CRC32	$7.02 \cdot 10^0$	FFT	Susan_e	FFT	FFT
MxM 128	$4.30 \cdot 10^0$	MxM 128	FFT	Susan_e	Susan_e
MxM 400	$1.65 \cdot 10^0$	CRC32	CRC32	CRC32	CRC32

of features for every architecture and code may not lead to suitable SDC FIT rate estimation. However, one may infer the best correction (i.e., set of features) for a target code by correlating it to a set of metrics that best reflect its behavior. For instance, one can classify the code into parallel or sequential (IPC), compute-intensive or memory-intensive (Cache utilization and variables lifetime), structured or unstructured data, and so forth. Further work is needed to explore additional possible features and to improve overall estimation, especially for more complex architectures such as A9.

Finally, we also studied the crash FIT rates applying the same methodology for SDCs, detailed in section III-D. The average accuracy of crash FIT rate estimation is about 3 to 6 orders of magnitude different for A5 and A9 respectively. Crashes, as pointed by other works [15], are mostly caused by faults in components not available to user-level software and are less dependent on which application is executing. Thus, while we can perform software fault injections in critical components for SDC events, we cannot do the same for crash events, nor derive insightful features to improve accuracy.

V. CONCLUSIONS

In this work, we have performed an extensive reliability evaluation of eight codes executed in two widely used ARM microprocessors, Cortex-A5 and Cortex-A9. We estimate the FIT rate using high-level software fault injection with a combination of architectural and code features. Then, we compare with expected FIT rates obtained from radiation experiments.

We show an average SDC FIT using the best estimation differs 20% overestimation on the A5 and 35% underestimation for the A9.

To improve the estimation accuracy we use four features for A5 and A9.

One can improve even further the accuracy using different features that may explain better the application behavior and the actual use of underlying hardware. Unfortunately, Crash rates are more challenging to estimate since crashes are mostly caused by components of the hardware which cannot be accessed by software.

Finally, this estimation method is much more accessible than radiation experiments and field tests, providing an inexpensive, fast, and more accurate FIT rate estimation. Once the device's technology factor is measured, which requires radiation experiments, one can readily estimate the FIT rate of several applications without the use of special facilities or extra hardware.

REFERENCES

[1] *Road vehicles — Functional safety*, ISO 26 262, 2015.

[2] R. Lucas, "Top ten exascale research challenges," Tech. Rep., February 2014. [Online]. Available: https://www.osti.gov/biblio/1222713

[3] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *International Journal of High Performance Computing Application*, vol. 28, no. 2, p. 129–173, May 2014.

[4] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "Demystifying soft error assessment strategies on arm cpus: Microarchitectural fault injection vs. neutron beam experiments," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2019, pp. 26–38.

[5] A. Vallero, S. Tselonis, D. Gizopoulos, and S. Di Carlo, "Multi-faceted microarchitecture level reliability characterization for nvidia and amd gpus," in *Proceedings of the IEEE 36th VLSI Test Symposium*, San Francisco, CA, USA, April 2018, pp. 186–191.

[6] F. F. d. Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, June 2019.

[7] D. Oliveira, L. Pilla, M. Hanzich, V. Fratin, F. Fernandes, C. Lunardi, J. Cela, P. Navaux, L. Carro, and P. Rech, "Radiation-Induced Error Criticality in Modern HPC Parallel Accelerators," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, February 2017, pp. 577–588.

[8] J. Ziegler and H. Puchner, *SER–history, Trends and Challenges: A Guide for Designing with Memory ICs*. Cypress, 2004. [Online]. Available: https://books.google.com.br/books?id=I_9tGwAACAAJ

[9] C. Slayman, "Jedec standards on measurement and reporting of alpha particle and terrestrial cosmic ray induced soft errors," in *Soft Errors in Modern Electronic Systems*. Boston, MA, USA: Springer US, 2011, vol. 41, ch. 3, pp. 55–76. [Online]. Available: https://doi.org/10.1007/978-1-4419-6993-4_3

[10] N. Mahatme, T. Jagannathan, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, "Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process," *Transactions on Nuclear Science*, vol. 58, no. 6, pp. 2719–2725, December 2011.

[11] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, September 2005.

[12] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in *Proceedings of the 37th annual international symposium on Computer architecture*, New York, NY, USA, 2010, pp. 461–472.

[13] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, and P. Alfke, "The rosetta experiment: atmospheric soft error rate testing in differing technology fpgas," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 317–328, September 2005.

[14] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2015, p. 297–310.

[15] D. Oliveira, L. Pilla, N. DeBardeleben, S. Blanchard, H. Quinn, I. Koren, P. Navaux, and P. Rech, "Experimental and analytical study of xeon phi reliability," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2017, Art. No. 28, pp. 1–12.

[16] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris, "Instruction-level impact analysis of low-level faults in a modern microprocessor controller," *IEEE Transactions on Computers*, vol. 60, no. 9, pp. 1260–1273, September 2011.

[17] X. Iturbe, B. Venu, and E. Ozer, "Soft error vulnerability assessment of the real-time safety-related ARM cortex-r5 CPU," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE, September 2016, pp. 91–96.

[18] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, November 2019, Art. No. 69, pp. 1–23.

[19] T. Santini, L. Carro, F. R. Wagner, and P. Rech, "Reliability analysis of operating systems and software stack for embedded systems," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2225–2232, August 2016.

[20] G. S. Rodrigues and F. L. Kastensmidt, "Soft error analysis at sequential and parallel applications in ARM cortex-a9 dual-core," in *Proceedings 17th Latin-American Test Symposium*. IEEE, April 2016, pp. 179–179.

[21] F. Rosa, F. Kastensmidt, R. Reis, and L. Ost, "A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, October 2015, pp. 211–215.

[22] A. Chatzidimitriou, M. Kaliorakis, S. Tselonis, and D. Gizopoulos, "Performance-aware reliability assessment of heterogeneous chips," in *Proceedings of the IEEE 35th VLSI Test Symposium*. IEEE, April 2017, pp. 109–115.

[23] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. D. Carlo, "RT level vs. microarchitecture-level reliability assessment: Case study on ARM(r) cortex(r)-a9 CPU," in *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, June 2017, pp. 117–121.

[24] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization*, December 2001, pp. 3–14.

[25] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," *Journal of Physics*, vol. 1021, pp. 12 037–12 041, May 2018.

[26] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*, Raleigh, NC, USA, February 2009, pp. 117–128.