# Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments

Athanasios Chatzidimitriou[†] Pablo Bodmann[*] George Papadimitriou[†] Dimitris Gizopoulos[†] Paolo Rech[*]
[†]*Dept. of Informatics and Telecommunications, University of Athens,* Athens, Greece
{achatz | georgepap | dgizop}@di.uoa.gr
[*]*PPGC, Institute of Informatics, Federal University of Rio Grande do Sul,* Porto Alegre, Brasil
{prjbodmann | prech}@inf.ufrgs.br

***Abstract*—Fault injection in *early* microarchitecture-level simulation CPU models and beam experiments on the *final* physical CPU chip are two established methodologies to access the soft error reliability of a microprocessor at different stages of its design flow. Beam experiments, on one hand, estimate the devices expected soft error rate in realistic physical conditions by exposing it to accelerated particles fluxes. Fault injection in microarchitectural models of the processor, on the other hand, provides deep insights on faults propagation through the entire system stack, including the operating system. Combining beam experiments and fault injection data can deliver deep insights about the devices expected reliability when deployed in the field. However, it is yet largely unclear if the fault injection error rates can be compared to those reported by beam experiments and how this comparison can lead to informed soft error protection decisions in early stages of the system design.**

**In this paper, we present and analyze data gathered with extensive beam experiments (on physical CPU hardware) and microarchitectural fault injections (on an equivalent CPU model on Gem5) performed with 13 different benchmarks executed on top of Linux on an ARM Cortex-A9 microprocessor. We combine experimental data that cover more than 2.9 million years of natural exposure with the result of more than 80,000 injections. We then compare the soft error rate estimations that are based on neutron beam and fault injection experiments. We show that, for most benchmarks, fault injection can be very accurately used to predict the Silent Data Corruptions (SDCs) rate and the Application Crash rate. The System Crash rate measured with beam experiments, however is much larger than the one estimated by fault injection due to unknown proprietary parts of the physical hardware platform that can't be modeled in the simulator. Overall, our analysis shows that the relative difference between the *total* error rates of the beam experiments and the fault injection experiments is limited within a narrow range of values and is always smaller than one order of magnitude. This narrow range of the expected failure rate of the CPU provides invaluable assistance to the designers in making effective soft error protection decisions in early design stages.**

***Index Terms*—CPU reliability, soft errors, failures in time, neutron beam, fault injection, microarchitecture simulation**

## I. Introduction

Reliability has become one of the main constraints for computing devices employed in several domains, from High Performance Computing (HPC) to automotive, military, and aerospace applications [1]–[3]. Reliability has been identified by the U.S. Department of Energy (DOE) as one of the ten major challenges for exascale performance computing [1]. In fact, a lack of understanding or the underestimation of devices and applications error rate may lead to lower scientific productivity of large scale HPC servers, resulting in significant monetary loss [4]. When the computing device is integrated in cyber-physical systems such as cars, airplanes, or Unmanned Aerial Vehicles (UAVs), high reliability becomes mandatory and unexpected errors should be strictly avoided.

While errors that may undermine the reliability of a computing system can come from a variety of sources such as environmental perturbations, manufacturing process, temperature, and voltage variations [5]–[7], we focus on radiation-induced effects since they have been found to be the main reliability threat in commercial devices [8]. Such soft errors may corrupt data values or logic operations and lead to Silent Data Corruption (SDC), crashes, or can be masked and cause no observable error [9]–[11].

To be useful and effective any method for the chip's reliability evaluation must be both fine grain (i.e., providing full visibility and understanding of faults effect in the microarchitecture and the software stacks) and realistic. A *fine grain* evaluation is employed to capture a clear understanding of the causes and effects of faults, to identify the most vulnerable parts of the hardware or software, and to observe how raw bit flips propagate through the system stack. A *realistic* evaluation ensures a correct prediction of the expected error rate of the device when used in the field and provides realistic models of the faulty behavior. The fine grain and realistic evaluation of the reliability of Commercial Off-The-Shelf (COTS) devices, that became increasingly common in both safety-critical and HPC applications, is challenging as information about the architecture and device characteristics is typically very limited.

Beam experiments are the most realistic way to measure the error rate of a code or device in conditions that are as close as possible to the actual ones after system deployment. However, beam experiments are coarse grain, i.e., errors can be observed only when they manifest at the application output or

when they compromise the system responsiveness. There is no information about the spatial/temporal location of the original fault and no information on its propagation pattern through the microarchitecture, the system software or the application software layers. It is, therefore, hard to identify the most vulnerable hardware resources or software code portions and to extend/generalize results to other applications or devices when relying on beam experiments only.

Fault injection on models of a microprocessor at different detail levels (architecture, microarchitecture, register-transfer, transistor) has been extensively used to evaluate the vulnerability of hardware architectures and software codes. By injecting faults in the hardware blocks of a microprocessor or particular parts of a software code it is possible to measure the probability for faults to impact the application output or the system responsiveness, i.e., the Architectural Vulnerability Factor, AVF [12]. Fault injection can then identify codes, code portions, or architectural resources which are more likely, once corrupted, to affect the system reliability. This information is extremely useful to design dedicated and efficient hardening solutions. It has been shown that fault injection at the microarchitecture level is a very effective (fast and accurate method) for early assessment of the reliability of a microprocessor [13], [14]. However, faults can typically be injected in a limited set of resources and, due to time constraints, simplified errors models (such as single bit flip) are usually adopted. In other words, if fault injection is not validated or tuned with physical experimental data, the efforts could potentially lead to imprecise results.

In this paper, we present data on 13 benchmarks executed on the top of Linux on a ARM Cortex-A9 system. The extensive beam experiment covers more than 2.9 million years of natural exposure. Additionally, we inject more than 80,000 faults. By *combining* beam experiments and microarchitectural fault injection results we provide a reliability evaluation which is both precise and realistic. Then, *comparing* the error rate predicted with beam experiments and fault injection, we evaluate at which level fault injection can be used to emulate beam experiment. To the best of our knowledge this is the first reported comparison and analysis of the two popular methods on top of an actual hardware and a detailed simulation model of a widely employed CPU.

As a case study we choose the ARM Cortex-A9 architecture as it was available in both a hardware platform (Xilinx Zynq ZedBoard) and microarchitecture-level model (in Gem5). Thanks to its high computing efficiency, ARM architecture is actually used in the next computational core of many supercomputers, including the new Sandia and Los Alamos National Lab. clusters [15]. For the same reasons, embedded architectures are attractive for safety-critical applications such as autonomous vehicles or space exploration systems [16].

We select a wide and heterogeneous set of codes from the MiBench suite [17]. The tested codes have different computing characteristics and stimulate different resources. This allows us to correlate the fault injection and beam experiment results with the code characteristics and to extend, under certain circumstances, the analysis to other algorithms with similar characteristics. The codes are executed on the top of Linux to simulate a realistic application and to show that our analysis can give deep information on complex systems reliability.

The main contributions of this paper are: (1) an extensive experimental evaluation of the reliability of ARM devices, based on beam experiments; (2) a fault injection detailed analysis of the vulnerability of codes executed on the top of Linux in ARM A9 devices; (3) the comparison and discussion of the error rate predicted through beam experiment and fault injection.

The rest of the paper is organized as follows. Section II discusses the background including details of the two reliability assessment methodologies. Section III reports related work from the literature to position our contributions. Section IV presents a detailed description of the evaluation methodologies (benchmarks, hardware, and software setups). In Section V we combine the results of the experimentation with the two setups and in Section VI we compare the findings and reports. Finally, Section VII draws the conclusions of the paper.

## II. Background

In this section we briefly review the basic concepts of radiation-induced effects in modern computing devices and the most commonly used methodoloies to evaluate their reliability.

### A. Radiation Effects in Electronic Devices

When a galactic cosmic ray interacts with the terrestrial atmosphere, it triggers a chain reaction that generates a flux of particles (mainly neutrons). About 13 $neutrons/((cm^2) \times h)$ reach ground [18]. A neutron strike may perturb a transistor's state, generating bit-flips in memory or current spikes in logic circuits that, if latched, lead to an error [19]. A transient error can have no effect on the program output (i.e., the fault is masked, or the corrupted data is not used) or propagate through the abstraction stack of the system leading to a Silent Data Corruption (SDC), or unrecoverable behaviors such as a program crash or device reboot.

The error rate of a code running on a computing device depends on both the memory/logic sensitivity [20], [21] and the probabilities for the fault to propagate through the architecture or program [12], [22]. Hardening solutions can be applied at different levels of abstraction (from transistor level to software or system level) to reduce the faults probability of occurrence or to avoid fault propagation.

### B. Reliability Evaluation Methodologies

The evaluation of the error rate of a device is essential to understand if the device meets the project's reliability requirement. Additionally, an early pre-silicon prediction of the device error rate is useful to evaluate if the reliability needs to be improved and to identify possible design vulnerabilities.

Realistic error rates can be measured exposing the real hardware to controlled particles beams. By exposing the device to particle beams, it is possible to mimic the effect of natural radiation on the final system in the field. Thanks to the high

TABLE I
PERFORMANCE OF DIFFERENT ABSTRACTION LAYER MODELS [23], [24].

| Abstraction Layer | Model | Performance (Cycles/sec) |
|---|---|---|
| Software (native) | Modern processors | $2 \times 10^9$ |
| Architecture | Gem5 atomic model | $2 \times 10^7$ |
| Microarchitecture | Gem5 detailed out-of-order model | $2 \times 10^5$ |
| RTL | NCSIM simulation | $6 \times 10^2$ |

particles flux intensity, a statistically significant amount of data is gathered in a short time. Moreover, the same kind of faults that would impact the device in its application in the field are injected in all physical hardware resources with realistic probabilities. Unfortunately, beam experiments offer limited visibility of fault propagation as faults are observed only when they compromise the system functionality (corrupting the output or crashing the application/system). With beam experiments it is then very hard to correlate the observed effects with their causes, limiting the identification of the system most vulnerable parts. Additionally, beam experiments can obviously be performed only on real hardware, after the device project has been finalized.

Designs that need to comply with certain dependability constraints require decisions to improve the reliability of the system but without adding unnecessary overhead. Reliability evaluation is intended also to support such decisions. It is critical to have this analysis in time and as early as possible, since any additional re-design iteration can lead to catastrophic costs. As a result, early-reliability assessment is often performed in models that exist prior to silicon prototypes, which can be summarized as architecture level, microarchitecture level and RTL. These vary in level of detail, with the most abstract being available earlier in the design chain while the most detailed (RTL) being available at the later stages. Architecture-level models often lack most, if not all, of the hardware details of the system, offering a software level functional emulation, while microarchitecture-level includes most functional and timing-accurate models of the microarchitecture (pipeline, cache memories etc.), offering clock cycle accuracy. In addition, most memory elements of the system (including SRAMs, pipeline registers and flops/latches not related to logic; e.g. state machines) are accurately modeled in microarchitecture level. RTL offers a full description of the implemented hardware, including the logic and SRAM components. Simulation time for each abstraction layer is proportional to the level of detail, with each detail step adding approximately 2 orders of magnitude more simulation time; the most detailed RTL model ends up being extremely slow. Table I illustrates the simulation throughput of each abstraction level.

Different reliability evaluation techniques can be applied on top of each model, delivering different levels of detail along with throughput. Probabilistic and statistical models [25], [26] often require a single simulation to deliver a rough estimation of the reliability, based on simulation statistics. ACE analysis [12], [27]–[29] on the other hand tries to capture more details on the residency and lifetime of workload critical data on each vulnerable component of the system, and weight their vulnerability against their sensitive exposure time. ACE analysis often requires one or a few simulations to quantify the vulnerability, but also requires additional development effort in order to capture all of the systems complexity. ACE analysis is claimed to have adjustable accuracy (proportional to the effort) but the tradeoff between effort and speed should always lean towards speed, otherwise more straight-forward approaches can be used [30]. Statistical fault-injection is one of the most widely adopted approaches of reliability assessment. It offers the flexibility of variable accuracy (depending on the size of statistical sample) while at the same time, it delivers failure samples produced by simulation. On the drawback side, the requirement of multiple simulations requires significant amount of time and, depending on the model detail, it can often be considered as unfeasible.

In this work we combine and compare the reliability evaluation performed using beam experiments and with statistical fault injection on top of microarchitecture-level models. We discuss at which level the pre-silicon reliability evaluation performed on microarchitectural models provides similar data when compared with beam experiments on real hardware.

Assuming fault injection and beam experiments are performed on exactly the same hardware, software, and OS configurations, there are still several reasons for beam and fault injection FIT rates not to be identical and for neither of them to be perfectly accurate if compared to the real device FIT rate. Figure 1 visualizes the different sources of uncertainties between the real FIT rate and the FIT rates measured with beam experiments or predicted with fault injection and shows the relative position of the reported FIT rates from each approach.

As some device structures cannot be modeled (e.g. logic-related latches), fault injection is likely to underestimate the device FIT rate. Additionally, a simplified fault model (typically a single bit flip model) is normally used for injections, which add additional uncertainty to the predicted error rates because in actual hardware implemented in recent technologies multiple bits may be flipped by a single particle strike.

When the real hardware is exposed to accelerated neutron flux, the whole chip is irradiated and a much more realistic behavior is modeled as neutrons hit the CPU chip. However, some resources/interfaces in the test board which are not part of the evaluated CPU are exposed to the beam and can be corrupted causing unresponsiveness in the system. Such cases can lead to an overestimation of the device under test FIT rate estimation through beam testing. In addition, the particles counts in the irradiation facility is not as precise as fault injection. This could lead to experimental errors that add uncertainty to beam results.

Finally, the real FIT rate of a device in the field depends on both the device sensitivity and the particle flux at which it is going to be exposed. Besides uncertainties in the sensitivity evaluations, also the particle flux is subject to significant

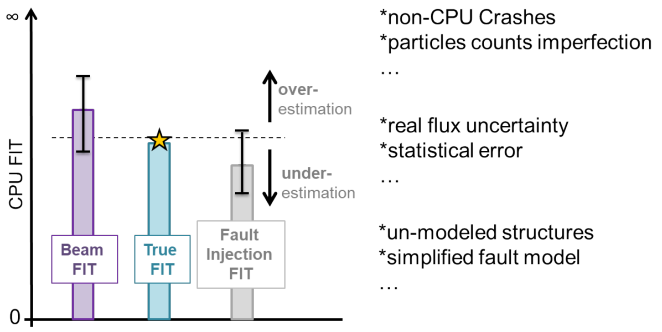


Fig. 1. Our motivation in comparing the different reliability evaluation methodologies and reasons for over- and under-estimations.

variations depending on the environmental conditions. JEDEC suggests to uses $13n/cm^2/h$ as flux at NYC [18], but it also states that the flux is dynamic and could significantly change over time. It is worth noting that field tests (i.e. exposing a high number of devices to the natural radiation) could potentially be more accurate than beam experiments and fault injection [31], [32]. However, a huge amount of devices and long time of exposure is required to gather a statistically significant amount of data, making field tests mostly unpractical.

In Section VI, as one of the main insights of our paper, we discuss the differences between the FIT rate we measure with beam experiments and microarchitectural fault injection and link the discussion to the abstract concept shown in Figure 1.

## III. Related Work

In this section we present related works in the field of reliability evaluation of modern devices through fault injection and accelerated beam experiments.

Particle accelerators have been used for many years to measure and study the reliability of devices and applications [20], [33]. Computing devices reliability has a strong tradition, motivated mainly by their use in safety-critical applications [9], [34], [35].

ARM Cortex-A9 processors have been exposed to accelerated particles beam and have been subjects and fault injection in previous studies. In [36], [37], [38], and [39] authors present beam experimental data on embedded ARM Cortex-A9, propose hardening solutions, and discuss the impact of the presence of an operating system in the application and device reliability. [40] and [41] present results on architecture-level fault injection of the processor core, while [42] includes a microarchitecture-level fault injection on A9. [24] presents a comparative reliability evaluation between microarchitecture and RTL fault injection, for baremetal workloads running on Cortex-A9, while [43], [44] also includes results of RTL fault injection on ARM CPU cores. Apart from ARM processors, fault injection on RTL was also used in [23], [28], [45].

Characterization of a full system using fault injection in architecture level is presented in [46]–[51] with some works only focusing on the vulnerability of the operating system. Some high-level fault injection tools have also been developed in the past that offer generic and cross-ISA software assessment [52], [53]. Microarchitecture level fault injection has been used in various studies [13], [54], [55] for assessing reliability on a hardware component basis, but also for capturing the performance deviation cause by the presence of faults in speculative components [56], [57]. There are also studies [58]–[63] that spread their focus beyond a single abstraction layer and aim to exploit the benefits of multiple abstraction layers to either accelerate the evaluation process, or deliver cross-layer reliability evaluation.

Some preliminary studies have proposed a comparison or combination of different reliability evaluation techniques [23], [28], [29], [39], [64], [65]. None of these works quantifies the accuracy of the fault injection reliability evaluation against the physical end product, considering the full system stack. This is the first paper that both compare and combine beam experiments and microarchitectural fault injection.

TABLE II
Summary of setup attributes.

| Property | Beam | Gem5 |
|---|---|---|
| Microarchitecture | Cortex-A9 | Cortex-A9* |
| Platform | Zynq 7000 | VExpress |
| CPU cores | 1* | 1 |
| L1 Cache | 32 KB 4-way | 32 KB 4-way |
| L2 Cache | 512 KB 8-way | 512 KB 8-way |
| Kernel version | 3.14 | 3.13 |

## IV. Methodology

In this section we present the evaluation methodology for the comprehensive comparative study of this paper. We first describe the ARM CPU and the selected benchmarks we execute on it. Then, we present the microarchitectural fault injection framework and the neutron beam experiment setup.

### A. Benchmarks and Devices

Our study is performed on an ARM®Cortex™-A9 architecture embedded in a Xilinx Zynq™-7000 AP System on Chip (SoC) implemented in a 28 nm CMOS technology and simulated in Gem5 (details in Section IV-C). The hardware device has two ARM cores operating at a maximum frequency of 667 MHz. Each core has a 32 KB 4-way set-associative instruction and data caches and a 512 KB 8-way set-associative Level 2 cache, which is shared between the cores [67]. We tuned the Gem5 model to resemble the physically available one and we have disabled the second core of the SoC in order to make the two evaluation setups as close as possible. The Linux kernel version we have tested on the Zynq board is 3.14 [68] while for Gem5 the used version is 3.13. These were the closest kernel versions that have been ported on the two platforms and were selected in order to minimize the operative system differences between the two setups. Table II presents the main characteristics of the two setups. Asterisks indicate two differences. First, Gem5 Resembles Cortex-A9 configuration, but the pipeline has some design differences and, second, in the Zynq the second core is still physically present, although disabled.

TABLE III
INPUT USED AND BENCHMARKS CHARACTERISTICS AS DESCRIBED IN [66].

| BENCHMARK | INPUT | CHARACTERISTICS |
|---|---|---|
| CRC32 | 26.6 MB file | CPU intensive |
| Dijkstra | 100x100 Integer Adjacency matrix | Control intensive, memory intensive |
| FFT | a single floating point array with 32768 elements | Memory intensive |
| Jpeg C | 512x512 PPM image with size of 786.5 KB | CPU intensive |
| Jpeg D | 512x512 PPM image with size of 786.5 KB | CPU intensive |
| MatMul | 128x128 Single precision floating point | Memory intensive |
| Qsort | a list of 50K doubles | Memory intensive and Control intensive |
| Rijndael E | 3.2 MB file | Memory intensive |
| Rijndael D | 3.2 MB file | Memory intensive |
| StringSearch | 1332 words to search in 1332 sentences (1 word per sentence) | Memory intensive and Control intensive |
| Susan C | 76x95 pixels, 7.3 KB | CPU intensive |
| Susan E | 76x95 pixels, 7.3 KB | CPU intensive |
| Susan S | 76x95 pixels, 7.3 KB | CPU intensive |

To have a broad analysis and avoid the bias of results on specific applications, we have chosen benchmarks with different computational characteristics. The applications chosen are part of *mibench* [66] testbench and are listed below. Table III shows the input used and some characteristics for each benchmark:

- CRC32: It calculates the corresponding 32-bit Cyclic Redundancy check (CRC) of a given file input. CRC is widely used in networks and storage devices in order to detect unwanted changes in the data.
- Dijkstra: This benchmark calculates the shortest path between 2 nodes using a adjacency matrix of size 100x100. 100 paths are calculated during each execution.
- FFT: It performs the Fast Fourier Transform (FFT) on a wave on a array of 32,768 floating point data. The FFT is widely used in digital signal processing.
- Jpeg C (Encode) and Jpeg D (Decode): These benchmarks convert one PPM image to jpeg format (Jpeg C) and vice versa (Jpeg D). The input file is the same 512x512 pixels image in two different formats: PPM for Jpeg C and jpeg for Jpeg D.
- MatMul: It multiplies two 128x128 matrices. This algorithm is used in image processing and Convolutional Neural Networks (CNN).
- Qsort: It sorts an array using the quick-sort algorithm implemented in the GNU C standard library. This algorithm was chosen in order to represent data sorting operations.
- Rijndael E (Encryption) and Rijndael D (Decryption): These two benchmarks use the Rijndael algorithm as defined in the Advanced Encryption Standard (AES). One encrypts an input file (E) and the other decrypts an encrypted input file (D).
- StringSearch: It searches a word in a sentence.
- Susan C: It uses the corner SUSAN algorithm in order to find the corners of the features. This algorithm is used to detect corners in features in an image.
- Susan E: It uses the edge SUSAN algorithm in order to find the edges of the features.
- Susan S: It uses the SUSAN algorithm in order to remove noise and preserve the image structure.

It is worth to mention that we use the exact same input vector, i.e., the same values and same size for each corresponding benchmark in both beam experiments and fault injection. Further discussion is presented in Section IV-D.

### B. Neutron Beam Experiments

Neutron beam experiments are one of the most precise ways to evaluate devices and applications error rates. Faults are induced with realistic probabilities and, as the whole chip is irradiated, faults are not restricted to a subset of accessible resources like for most software fault-injection frameworks. Our radiation experiments were performed at the LANSCE facility of the Los Alamos National Laboratory (LANL) in Los Alamos, NM.

Figure 2 shows part of our setup at LANSCE. We irradiate four Xilinx Zedboards with a $2 \times 2$ inches beam spot, which is sufficient to irradiate the chip uniformly without affecting the onboard DDR. This means that data in the DDR is not expected to be affected by radiation.

During the experiment, the ARM output is compared with a golden reference which contains the expected output (precomputed in a fault free environment). Any mismatch between the experimental and expected output is marked as an **SDC** and logged for later analysis. Additionally, during the execution the ARM sends an "Alive" message to a host PC to indicate the correct function of the application. If after a given period of time no message is received, an attempt is made to contact to the board and restart the application. If the attempt is successful, the event is logged as an **Application Crash** (Linux is still running and responding). If no connection with the board can be establish, the event is logged as a **System Crash**, as the operating system has crashed.

LANSCE provides a neutron beam suitable to mimic the atmospheric neutron effects in electronic devices. The available neutron flux was about $3.5 \times 10^5 n/(cm^2/s)$, about 8 orders of magnitude higher than the terrestrial flux ($13n/(cm^2 \times h)$ at sea level [18]). Our experiment ran for about 260 effective beam hours (i.e., not considering setup, initialization, and recover from crash times), which, when scaled to the natural exposure, account for more than 2.9 million years. Since the terrestrial neutron flux is low, in a realistic application

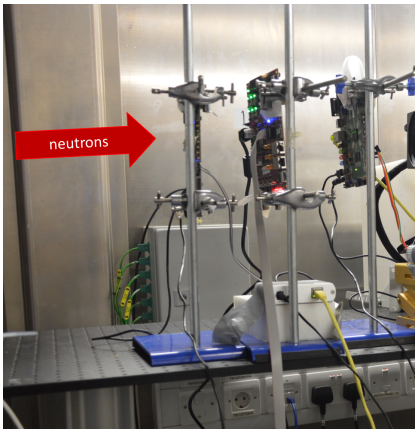


Fig. 2. Radiation test setup at LANSCE.

it is highly unlikely to see more than a single corruption during program execution. We have carefully designed the experiments to maintain this property (observed error rates were lower than 1 error per 1,000 executions). Experimental data, then, can be scaled to the natural radioactive environment without introducing artifacts.

Experiments aim to measure the **Failures In Time (FIT)** rate of the device executing a code (failures per $10^9$ hours of operation). FIT depends only on the kind and amount of resources required for computation, without considering the code execution time [20].

### C. Fault Injection

The microarchitectural modeling is based on Gem5 simulator, a flexible full system cycle-accurate simulator [69]. Gem5 fully supports ARM ISA and comes along with a detailed out-of-order core implementation. The CPU core was configured to resemble the microarchitecture of Cortex-A9. Among the various abstraction models available, microarchitecture level is the only one that comes with sufficient hardware detail and is capable of running full-system simulation. While RTL is certainly more accurate, the limited simulation throughput does not allow running of multiple fault injection simulations of a full system stack that includes an operating system and transactions with I/O peripheral devices.

GeFIN fault injection framework [13] was used on top of Gem5 for the reliability assessment. In order to replicate the effects of beaming, GeFIN was configured to inject single-event transient faults during the simulation of the system. The faults were injected in six components: L2 Cache, L1 Data and Instruction Caches, Physical Register file, Data and Instruction Translation Lookaside Buffers (TLB). These components cover more than 94% of the available memory cells modeled inside the CPU.

A set of 1,000 single-bit faults was generated for every component, resulting in 6,000 fault injection simulations per program. According to [70] formulation on statistical fault sampling calculation, this corresponds to 4% error margin with 99% confidence level. This estimation corresponds to an initial unknown AVF estimation [12], which, as suggested by [70], is set to p = 0.5 in order to maximize the fault sample. After the execution of simulation campaign however we can re-adjust the p variable in the formula with the result of the estimation, shifted by the maximum error margin. This gives us a tighter estimation of the error margin for each combination of workload/component, which in our results varies between 1.7% and 4% with 99% confidence. Table IV shows the error margin range for each component in the fault injection campaigns.

TABLE IV
MIN, MAX, AND AVERAGE ERROR MARGIN FOR EACH COMPONENT ACROSS WORKLOADS FOR A GIVEN FAULT SAMPLE OF 1,000 FAULTS.

| Component | Min Err | Max Err | Avg Err |
|---|---|---|---|
| Register File | 2.2 % | 3.3 % | 2.9 % |
| I$ Cache | 2.6 % | 3.7 % | 3.0 % |
| D$ Cache | 2.4 % | 4.0 % | 3.7 % |
| L2 Cache | 1.7 % | 4.0 % | 3.7 % |
| DTLB | 3.7 % | 4.0 % | 4.0 % |
| ITLB | 3.8 % | 4.0 % | 4.0 % |

Microarchitecture level fault injection offers significant amount of observability, allowing distinction of where exactly did the fault strike (e.g., whether it was on kernel or user mode or data, whether the corrupted entry was used or not etc.) but also detailed information of what was the system effect. The default fault classification is sufficient to match the beam experiment fault effect classes, and each fault injection simulation can be characterized as SDC, System Crash, Application Crash or Masked, depending on the result of the simulation. That being said, the vulnerability estimation of GeFIN can be compared against beam experiments, but requires a conversion, as the original outcome of a fault injection campaign is vulnerability estimation [12] rather than failure rate. Details on how AVF can be attributed to FIT rates are presented in Section VI.

### D. Fault Injection vs. Radiation Experiment

To avoid any difference not related to the reliability evaluation that can bias our results we used exactly the same source code, compiler, compiler options, and input vector (size and values) for both fault injection and beam experiments. Still, the setups used for beam experiments and fault injection are intrinsically different as we are comparing the execution on actual hardware vs. Gem5 simulation [71]. To evaluate if the benchmark execution shows any difference when running in the setups used for beam experiments and fault injection, we compare the execution of the same code in the two setups using 7 different counters: CPU cycles, branch misses, L1 data cache accesses, L1 data cache misses, L1 data TLB misses, L1 instruction cache misses, and L1 Instruction TLB misses. About 70% of the counters report acceptable deviations between the two setups. The biggest difference is observed in the L1 Instruction TLB counters. Literature actually identifies certain design differences that exist in the implementation of TLB of Gem5 and ARM Cortex microarchitectures that support these observations [71]. Differences are to be expected

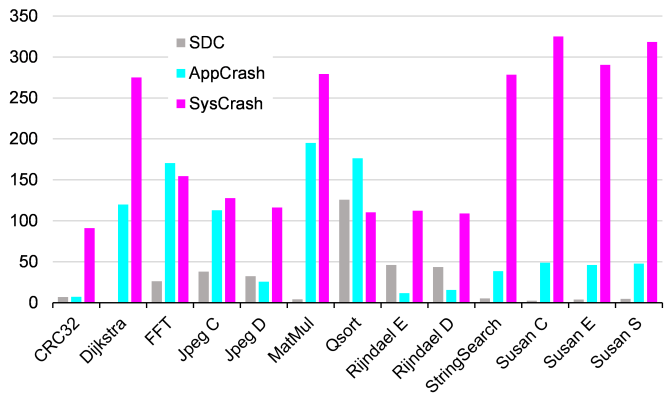


Fig. 3. Beam FIT rates for SDCs, Application Crashes and System Crashes.

as the Gem5 model is not exactly the same as the one implemented in hardware. The main goal of this paper is to understand if microarchitectural simulations can provide accurate insights on the corresponding hardware reliability.

To predict the error rate of a code using fault injection so to compare it with the one measured with beam experiments it is necessary to know the raw probability for faults to occur. In principle, multiplying the raw fault probability of each microarchitectural resource by its AVF would provide the realistic error rate of the code executed on the device. However, measuring the fault probability for each hardware resource would require too much time and, when dealing with COTS, could be unfeasible due to visibility limitations. In this paper, we decide to use the experimentally measured raw fault rate of a bit in the L1 cache as a reference for the Cortex-A9 technology fault probability. This simplification is justified by the fact that caches are normally the most vulnerable resource in a microprocessor and also the targeted components in GeFIN are implemented all in the same SRAM technology as the L1 cache.

## V. Fault Injection and Beam Testing Data

In this Section we present and discuss beam experiment results and fault injection analysis, highlighting the main insights on the reliability of the ARM A9 each methodology provides. In the next Section we will compare the FIT rates measured with beam experiments and predicted with fault injection.

### A. Beam Experiments Results

Figure 3 shows the FIT rates for the 13 benchmarks tested at LANSCE following the experimental procedures detailed in Section IV-B. We report the FIT rate for SDC, Application Crash, and System Crash. It is worth noting that these three types of events are uncorrelated and independent, in the sense that at most one of the three events occurs in one execution and its occurrence will not affect the probability of errors in the next executions.

From Figure 3 it is clear that a **System Crash** is the most likely event, for all the benchmarks but FFT and Qsort. As discussed in the following, FFT and Qsort have a higher Application Crash FIT compared to other benchmarks; their Application Crash FIT is even higher than their System Crash rate. As shown in [39], System Crashes have a component which is intrinsic to the particular hardware platform, only. Even resilient codes (i.e., with very low SDC or Application Crash rates), then, could experience a relative high number of System Crashes. This is the case of CRC32, Rijndael D, and Rijndael E in Figure 3. The benchmarks with the highest System Crashes FIT are Dijkstra, MatMul, StringSearch, and the three Susans benchmarks. These are actually the benchmarks with the smallest input size (please refer to Table III), which is not even sufficient to fill all caches. Since there is space available in the caches, the Linux kernel code will not be evicted when the context is switched to the application. When Linux is in idle, its data is then exposed and vulnerable to radiation. An error in the kernel code is likely to lead to a System Crash, exacerbating the System Crash vulnerability. This is aligned with previous studies that shown a higher operating system error rate in the absence of cache conflicts [36].

There is also a significant variation among the **Application Crash** FIT of the different benchmarks. The benchmarks with highest Application Crash rate are MatMul, Qsort, and FFT while the lowest (about 2 orders of magnitude lower than MatMul) is CRC32. The codes with the higher Application Crash FIT are found to be the ones with higher presence of control-flow operations, higher use of stack for memory or nested loops. Additionally, these applications are the ones with non-coherent memory accesses. This requires several accesses (reads and writes) to the main memory, which is outside the irradiated chip. To access the main memory it is necessary to use an interface between the core and the main memory. It has already been demonstrated that errors during intra-chip communications are likely to lead to an Application Crash as an error in the interface or in one of the cores involved in the communication makes the application to wait indefinitely [39]. We observe as well that while Rijndael has similar Application Crash FIT rate for both encoding and decoding, for Jpeg C the coding has a much higher (almost 1 order of magnitude) Application Crash FIT rate than the decoding. This can be explained by the fact that in the case of Rijndael the process to encode and decode is algorithimcally almost identical, but in Jpeg D the decoding is achieved by doing the reverse steps from the encoding. This means that the program flow is different and behaves differently when an error occurs.

For **SDC** FIT rate, the difference between the lowest (Dijkstra) and the highest (Qsort) is around 2 orders of magnitude. This result confirms previous studies showing that the SDC rate is strongly application dependent [39]. The codes with lower SDC FIT rate are the three Susans, StringSearch, MatMul, Dijkstra, and CRC32. These benchmarks either have a small input (Susans, StringSearch, MatMul) or have long memory latency (CRC32). Long memory latency, in fact, has been shown to reduce the error rate of a device as while waiting for memory transfer the core is in idle state and, thus, unlike to produce an error [39]. We also observe that for SDC

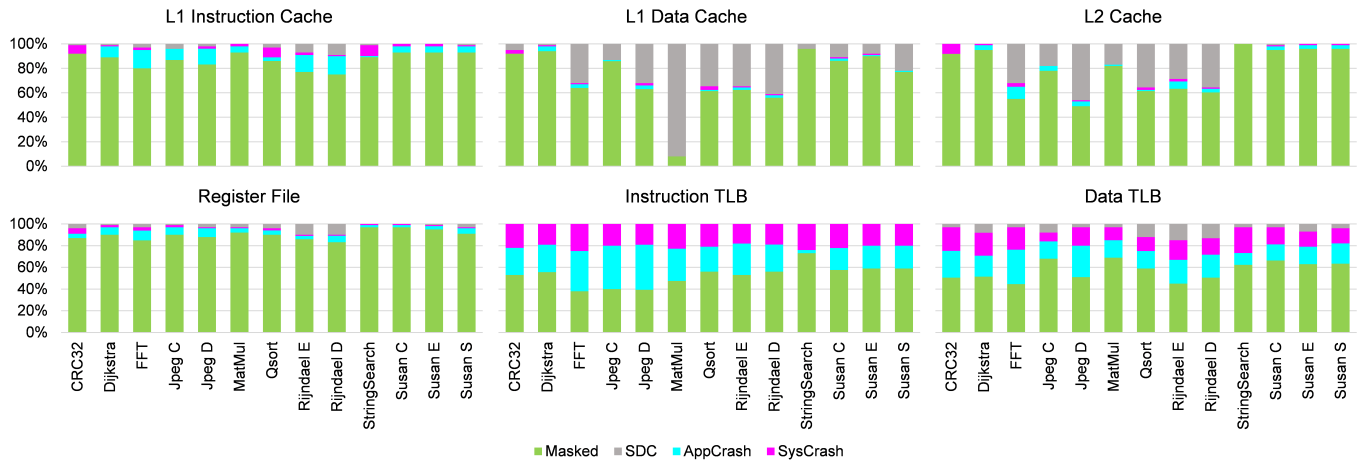


Fig. 4. Fault injection effects classification for all 13 benchmarks in all 6 components. Effects are Masked, AppCrash, SysCrash, and SDC. The AVF corresponds to the sum of the three non-Masked cases.

rate for both encoding and decoding of Jpeg and Rijndael benchmarks have similar FIT SDC rate. This can be explained noting that, for Rijndael, both encoding and decoding have equal input size and, for Jpeg, the combination of input and output have equal size (the input of the encoding and the output of the decoding have almost identical sizes).

### B. Fault injection results using GeFIN

Unlike beam experiments, fault injection campaigns were executed for each hardware component separately. Figure 4 illustrates the AVF estimation reported by GeFIN. With fault injection we can also count the faults that were benign (masked) and the fault was either overwritten or did not affect the execution in any observable way. Figure 4 presents the AVF distribution of each fault class, SDC, Application Crash or System Crash, while the summary of all non-masked cases expresses the vulnerability of the structure (AVF). Notice that AVF does not only depend on a structure's size or organization, and is aggregately related to the significance of the stored data to the correct operation of the system.

As one would intuitively expect, the majority of SDC results (grey areas) are related with the structures that mostly contain data, which are the L1 Data Cache and L2 Cache. In contrast, we can see how most of the abnormalities reported by faults in the L1 Instruction Cache are crashes. Interestingly, we can see that most of the benchmarks have higher Application Crashes than System Crashes, with CRC32, Qsort and StringSearch being the only outliers.

The TLBs are consistently highly vulnerable. The reported fault injections refer to the Physical page (target) of the tables as they mostly lead to either incorrect memory translations or wrong permission flags. Incorrect translations will lead to use of wrong data in all references of the particular page. In contrast, the virtual part (tag) has almost zero vulnerability as corruption in the tag can mainly result to tag misses and thus invoke unnecessary page walks, which introduces a performance penalty. The Register file is involved in both control and data processing and the vulnerability is evenly distributed in all classes, without particular trends. However, we can see how both Rijndael benchmarks report high probability of SDCs, and this can be attributed to the high level of instruction level parallelism of the algorithms, which results in high utilization of the register file for data processing.

Although the AVF measurement is not related only to the size of a component, the probability that a fault is introduced at a particular structure (by a particle) is highly related to its size. Each TLB for instance, has a size of 512 bytes (4,096 bits) while the L1 Cache memories have a size of 32KB (252,144 bits). Consequently, the probability that a fault will strike the TLB is only $1/64^{th}$ of the Cache's probability. That being said, the L2 Cache, which covers more than 80% of the modeled memory cells of the system, suffers the most by the striking of faults.

## VI. Comparison of Fault Injection and Beam Data

In this Section we compare the FIT rate measured with beam experiments and predicted with fault injection simulations.

The Architectural Vulnerability Factor expresses the probability that a fault in a specific hardware structure can result in a corruption in the execution of a program. The metric is independent of the components technology or environmental factors, which are however related to the soft error rate. There is a direct connection between the failure rate and the vulnerability of a structure. The soft error rate quantifies how many faults are introduced in a hardware structure at a given period of time. As the structures size increases, the error rate is also increased. If we want to attribute the FIT of a component to its size, we need to have a per-bit FIT, which is called raw FIT, or $FIT_{raw}$. The FIT of a component can then be expressed as:

$$FIT_{component} = FIT_{raw}(bit) * Size(bits) * AVF_component$$

The size of each component is known a priori and the AVF is provided by GeFIN. The only missing attribute is the

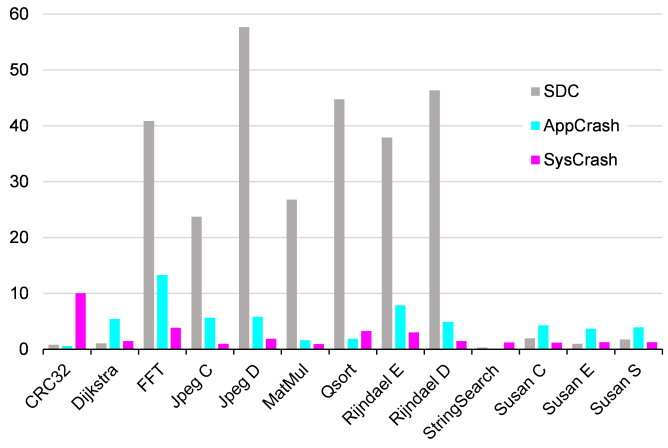


Fig. 5. Fault Injection FIT rates for SDCs, Application Crashes and System Crashes.

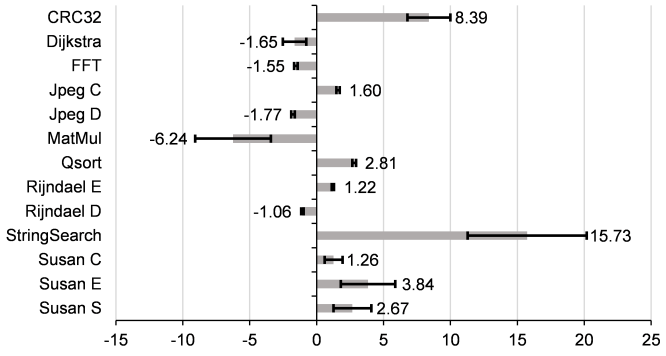


Fig. 6. SDC FIT comparison between radiation experiments and fault injection.

one that depends on the technology, which is the $FIT_{raw}$. To have a perfect emulation of the realistic error rate we would need the $FIT_{raw}$ for all the resources in the processor, which is unfeasible mainly for COTS devices, as explained in Section II. We decide to use the L1 cache bit $FIT_{raw}$ as representative of the Cortex-A9 technology as implemented in the Xilinx Zynq. We choose L1 cache because it is among the most vulnerable resources, while at the same time uses the same SRAM technology with rest of the CPU components. The L1 cache bit FIT was used as a common baseline for all the resources in the ARM CPU.

To experimentally measure the $FIT_{raw}$, we exposed a specific benchmark designed to test the L1 data cache. This is achieved filling byte-by-byte the L1 data cache with a known pattern and read it after a period of time, comparing the read values with the pattern. This gives us the FIT rate for the cache and, diving it by the tested cache size, it gives us the cache FIT per bit. For the L1 cache, the measured error rate is $2.76x10^{-5}$ FIT per bit, very close to other publicly available data for the same technology [72].

Using the $FIT_{raw}$ we can predict the FIT rate of applications based on the AVF analysis, as shown in Figure 5. In order to compare the FIT rate predicted with beam and fault injection, for each code we divide the highest FIT rate between the one calculated with beam data and the one predicted using fault injection by the lowest FIT rate between the two. Whenever the fault injection FIT rate is higher than the beam one we represent the value as positive, negative otherwise.

Figure 6 shows the comparison between beam and fault injection SDC FIT rates for all 13 benchmarks. The positive values (towards the right) of the horizontal axis indicate how many times the FIT rate measured with beam experiments is higher than the FIT rate calculated with fault injection while negative numbers indicate the opposite (fault injection FIT rates are higher). For most benchmarks radiation and fault injection give very close FIT rates (for 10 out of 13 codes the difference is smaller than 4x, while for 7 of them it is less than 2x). MatMul, StringSearch, and CRC32 have the largest difference between the two FIT rate measurements. However, these benchmarks have very low SDC FIT rate, for instance, StringSearch has 5.45 SDC FIT on the radiation experiment and only 0.34 on the fault injection, meaning that the absolute difference between FIT rates is very small and such differences are within the statistical error. As expected from the discussion in Section II, for most of the benchmarks the FIT rate measured with beam experiment is higher than the fault injection one. However, we observe that for 5 benchmarks (Rijndael, Jpeg D, FFT, Dijkstra, and, mainly, MatMul), fault injection reports a higher SDC FIT rate than beam experiments. As discussed in the following sections, these are also the benchmarks that show a much higher Application Crash and System Crash rate for beam experiments compared to fault injection and this difference implies that some faults propagate differently to generate SDCs or Crashes in the two setups but still result in a corruption of the correct execution.

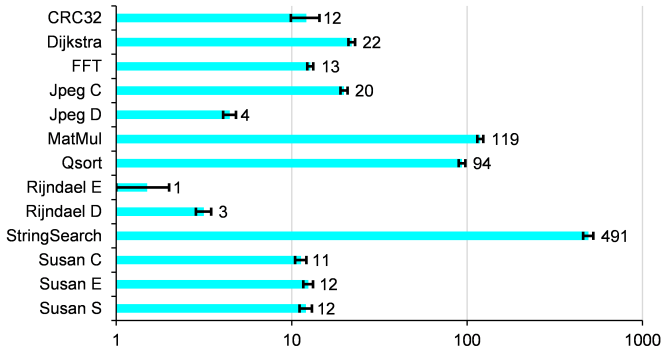


Fig. 7. Application Crash FIT comparison between radiation experiments and fault injection.

When comparing the Application Crash FIT rate calculated with the two setups, as shown in Figure 7, we observe that the differences between fault injection and beam experiments are much higher than for SDCs, ranging from 1.5x to almost 500x (horizontal axis is in logarithmic scale). It is worth noting that beam experiments FIT is always higher than the fault injection

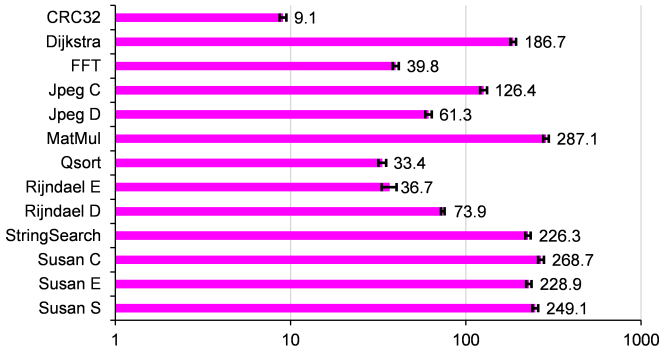


Fig. 8. System crash FIT comparison between radiation experiments and fault injection.

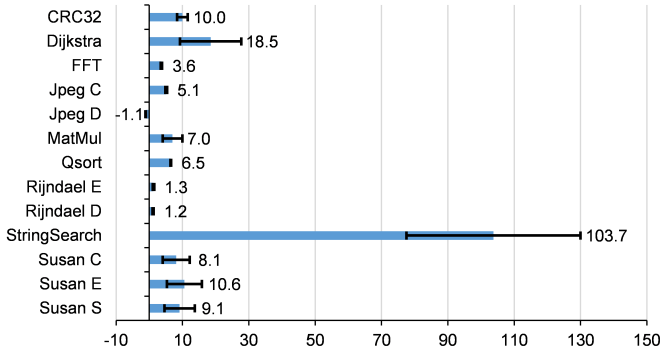


Fig. 9. SDC and Application Crash Comparison between radiation experiments and fault injection.

estimation, which is expected since Application Crashes could be triggered by corruption in logic/control hardware elements which are difficult to simulate [39]. For three benchmarks, StringSearch, MatMul, and Qsort, the difference between beam experiments and fault injection is close or bigger than two orders of magnitude (while for all others the difference is smaller than 22x). The reason behind this may be attributed to differences between the two setups. While fault injection experiments output is downloaded and compared off-line against the fault-free output to detect SDCs, beam experiments require an additional routine for on-line SDC checking. As during beam experiments most executions are error-free, downloading all outputs would be an unnecessary waste of space and time. These checks are almost transparent to the workload characteristics and, to avoid the corruption of SDC details, they are intentionally designed to hold pointer references instead of actual data. Application Crashes are mostly sourcing in abnormalities caused in the program flow (i.e., irregular branches, wrong memory references, etc.). These have roots in the executed code of a program (of what is placed in the .text section of a program). The common property of the 3 workloads with greater differences between fault injection and beam data (StringSearch, MatMul, and Qsort) is the relatively small code size, which fits inside the Instruction Cache. As a result, the code sits in the cache for the whole experiment, being exposed to neutrons. Additionally, there is enough space for the SDC check routines to remain in the cache hierarchy, instead of being evicted during the program execution (as L2 is shared) during the beam experiments. We believe that the exposure of these routines to the beam (which mainly consist of pointers) would result in segmentation faults that translate to Application Crashes. This is inevitable difference between the two setups could explain the observed behavior for the three outliers.

The System Crash FIT difference, as shown in Figure 8, does not follow the same behavior of the Application Crash FIT. We observe a high difference between radiation and fault injection for all benchmarks, with the radiation FIT being always higher than the injection. The difference ranges from about 9 times (CRC32) to about 287 times (MatMul). The benchmarks with the largest difference are MatMul, Dijkstra, StringSearch, and the three Susans. These workloads also happen to have the smallest inputs. As a result, they actually leave a large part of the cache hierarchy unused. We believe that the observed differences can again be induced by differences on the setups. In fault injection experiments, this portion remains empty as the caches are reset on every experiment, while in radiation experiments this space is used by the kernel for other system operations (e.g. scheduling routines, timer handlers etc.). The introduction of faults in these regions that will most likely result to system crashes only in the radiation experiments. The rest of the benchmarks that use most of the cache hierarchy do evict the kernel from the caches and do not suffer from this scenario.

As we have shown in Figures 6, 7, and 8, the magnitude of the differences between beam experiment and fault injection FIT rate are application dependent while there exists a clear trend of larger FIT rates measured by beam experiment compared to fault injection for all FIT rates (SDC, Application Crash, and System Crash FIT rates). As discussed in Section II, there are various reasons for fault injection and beam experiments to provide different FIT rates and this is the reason why they have to be considered complementary to each other. One very likely reason for the differences and particularly of the larger number of corruptions in the beam experiments are the resources that are not modeled in the simulator. We believe that this high System Crash FIT rate is a peculiar characteristic of the Xilinx Zynq platform, specifically the FPGA-ARM interface based on interrupts, which cannot be further investigate without detailed (proprietary) information.

An indirect way to correlate the results and focus on the same hardware is by attributing the effects that different hardware parts cause. While System Crashes, exceptions and wrong memory accesses can be caused by most of the system's components (including CPU cores, peripherals, controllers, bridges and interconnections), SDCs can only occur at the components that produce the output, which is the CPU core. This also partially applies to the majority of Application Crashes. In Figure 9 we plot the relative difference of the *sum* of the FIT rates for SDCs and Application Crashes measured

with beam and fault injection. StringSearch has the highest relative difference (of about 100x). This is due to the *extremely* small number of SDCs observed in both setups, having much less events observed in the injection than in the radiation setup. It is also interesting that the MatMul and Qsort, that show a difference of 100 times in Application Crash FIT (Figure 7), now have a difference lower than 10 times when comparing SDC and Application Crash FIT rates (Figure 9). This means that, the overall FIT rate is only 10x higher in the beam case. It is likely that some of the Application Crashes observed in the radiation experiment are observed as SDCs in the injection. This is probably caused by the corruption of some hardware resource not modeled in the injection setup. The other benchmarks are less affected by the code characteristics discussed previously. For three benchmarks, Jpeg D and the two Rijndael, the overall FIT difference is very small, from 1.08x up to 1.26x.

Figure 10 shows an aggregate view of our measurements. It presents the differences between the average FIT rate of the 13 benchmarks measured with beam experiment and fault injection compared to the (unknown) real FIT rate of an ARM Cortex-A9 CPU in the field. As expected and discussed in Section II, fault injection tends to report smaller device FIT rates than beam experiments. For SDC rates (leftmost bars) we can claim that both beam experiments and fault injection provides, on the average, very similar FIT prediction. It is reasonable to believe that the real SDC FIT rate of the device lays between the two values. When Crashes are considered (especially System Crashes - rightmost bars), the difference between beam and fault injection increases. As discussed, this is mainly due to un-modelled structures in fault injection and to the fact that a full system is massively irradiated with the beam. However, still the FIT rates including Crashes have a difference which is smaller or in the worst case equal to an order of magnitude (in our case the FIT rates difference when Application Crashes are added to SDCs is only 4.3x while the total FIT rate when Application and System Crashes are added to the SDCs - Total FIT in the rightmost bars - is only 10.9x). Again, based on our analysis, we can claim that the real FIT rate of the evaluated CPU may sit between the FIT rates values provided by beam experiments and fault injection and this rather narrow range can drive early informed decisions by the chip designers about soft error protection techniques for a particular CPU.

## VII. Conclusions

We presented the first detailed analysis that aims to report a head-to-head comparison of two very popular reliability assessment methods: (a) physical accelerated beam test of an ARM Cortex-A9 CPU and (b) fault injection on the corresponding model of the ARM Cortex-A9 CPU on the state-of-the-art microarchitectural simulator Gem5. For an as-close-as-possible comparison, we maximize the equivalence of the physical system setup and the simulated system setup: hardware configuration, application software, and operating system.

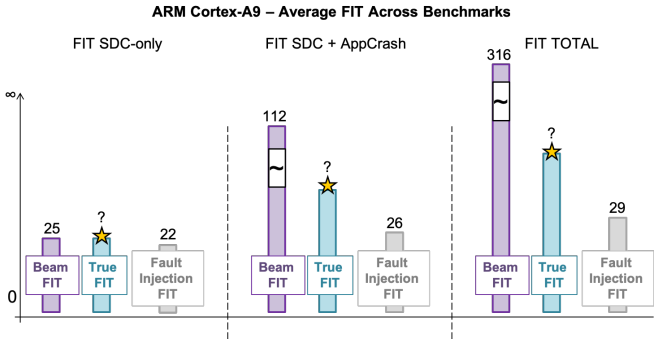


Fig. 10. Overview of the comparison between beam and fault injection FIT rates (compare with the motivation of the paper shown in Figure 1). Fault injection average FIT rate (grey bars) is dominated by the SDC FIT rate (leftmost bars) and is only slightly increased when the Application Crashes are added (middle bars) or the System Crashes are also added (rightmost bars). On the contrary, the Beam FIT rates (purple bars) are increased when the two types of Crashes are added. However, the SDC FIT rate of Beam is very close to the Fault Injection SDC FIT rate and also the differences when one or both types of Crashes are added are close to only one order of magnitude.

The comparison of the two reliability assessment approaches helps in bounding the range of the expected FIT rates of a CPU when it is deployed in a final system in the field. We have shown that for the diverse set of benchmarks employed in our experiments, the FIT rates differences between accelerated beam test and microarchitectural fault injection can be extremely small (when only the SDC FIT rate is considered) and does not exceed one order of magnitude when all types of errors (including Application and System Crashes) are considered for the Total FIT rate of the system. The insights of our study can assist CPU designers in making informed decisions about the soft error protection mechanisms best suited to a particular hardware and software combination.

## Acknowledgment

This work is partially funded by the H2020 Framework Program of the European Union through the UniServer Project, under Grant Agreement 688540, by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404, by the Coordenao de Aperfeioamento de Pessoal de Nvel Superior - Brasil (CAPES) - Finance Code 001, and by the project FAPERGS 17/2551-0001 202-0.

## References

[1] R. Lucas, "Top ten exascale research challenges," in *DOE ASCAC Subcommittee Report*, 2014.

[2] J. Dongarra, H. Meuer, and E. Strohmaier, "ISO26262 Standard," 2015. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en

[3] A. Cohen, X. Shen, J. Torrellas, J. Tuck, Y. Zhou, S. Adve, I. Akturk, S. Bagchi, R. Balasubramonian, R. Barik, M. Beck, R. Bodik, A. Butt, L. Ceze, H. Chen, Y. Chen, T. Chilimbi, M. Christodorescu, J. Criswell, C. Ding, Y. Ding, S. Dwarkadas, E. Elmroth, P. Gibbons, X. Guo, R. Gupta, G. Heiser, H. Hoffman, J. Huang, H. Hunter, J. Kim, S. King, J. Larus, C. Liu, S. Lu, B. Lucia, S. Maleki, S. Mazumdar, I. Neamtiu, K. Pingali, P. Rech, M. Scott, Y. Solihin, D. Song, J. Szefer, D. Tsafrir, B. Urgaonkar, M. Wolf, Y. Xie, J. Zhao, L. Zhong, and Y. Zhu, "Interdisciplinary research challenges in computer systems for the 2020s," USA, Tech. Rep., 2018.

[4] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson *et al.*, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, pp. 1–45, 2014.

[5] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, Jan 1993, pp. 126–133.

[6] J. C. Laprie, "Dependable computing and fault tolerance : Concepts and terminology," in *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on*, Jun 1995, pp. 2–.

[7] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*, 1999, pp. 86–94.

[8] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005.

[9] C. Constantinescu, "Impact of deep submicron technology on dependability of vlsi circuits," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 205–209.

[10] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, no. 6, pp. 30–39, Nov 2005.

[11] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: A large-scale field study," in *SIGMETRICS*, 2009.

[12] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 29–.

[13] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr 2016. [Online]. Available: https://doi.org/10.1109/ispass.2016.7482075

[14] M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez, "MeRLiN," in *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA '17*. ACM Press, 2017. [Online]. Available: https://doi.org/10.1145/3079856.3080225

[15] N. Hemsoth. (2018) Arm it the nnsas new secret weapon. [Online]. Available: https://www.nextplatform.com/2018/11/07/arm-is-the-nnsas-new-secret-weapon/

[16] NASA, "Phonesat project," https://www.nasa.gov/content/phonesat/, 2016.

[17] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[18] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," JEDEC Standard, Tech. Rep. JESD89A, 2006.

[19] 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," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 6, pp. 2719–2725, 2011.

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

[21] J. Noh, V. Correas, S. Lee, J. Jeon, I. Nofal, J. Cerba, H. Belhaddad, D. Alexandrescu, Y. Lee, and S. Kwon, "Study of neutron soft error rate (ser) sensitivity: Investigation of upset mechanisms by comparative simulation of finfet and planar mosfet srams," *Nuclear Science, IEEE Transactions on*, vol. 62, no. 4, pp. 1642–1649, Aug 2015.

[22] 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*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 461–472. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816023

[23] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. ACM Press, 2013. [Online]. Available: https://doi.org/10.1145/2463209.2488859

[24] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. D. Carlo, "RT level vs. microarchitecture-level reliability assessment: Case study on ARM(r) cortex(r)-a9 CPU," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, Jun 2017. [Online]. Available: https://doi.org/10.1109/dsn-w.2017.16

[25] G.-H. Asadi *et al.*, "Balancing performance and reliability in the memory hierarchy," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, ser. ISPASS '05. Washington, DC, USA: IEEE Computer Society, 2005.

[26] J. Suh, M. Annavaram, and M. Dubois, "MACAU: A markov model for reliability evaluations of caches under single-bit and multi-bit upsets," in *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, Feb 2012. [Online]. Available: https://doi.org/10.1109/hpca.2012.6168940

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

[28] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, p. 460, Jun 2007. [Online]. Available: https://doi.org/10.1145/1273440.1250719

[29] N. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and AVF estimation revisited," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, Jun 2010. [Online]. Available: https://doi.org/10.1109/dsn.2010.5544276

[30] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee, "Computing accurate AVFs using ACE analysis on performance models: A rebuttal," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 21–24, Jan 2008. [Online]. Available: https://doi.org/10.1109/l-ca.2007.19

[31] 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, Sept 2005.

[32] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: positional effects in dram and sram faults," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 22.

[33] J. F. Ziegler and H. Puchner, *SER–history, Trends and Challenges: A Guide for Designing with Memory ICs*. Cypress, 2010.

[34] N. Seifert, X. Zhu, and L. W. Massengill, "Impact of scaling on soft-error rates in commercial microprocessors," *Nuclear Science, IEEE Transactions on*, vol. 49, no. 6, pp. 3100–3106, 2002.

[35] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 365–381, Sept 2005.

[36] 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, Aug 2016.

[37] A. B. de Oliveira, G. S. Rodrigues, and F. L. Kastensmidt, "Analyzing lockstep dual-core arm cortex-a9 soft error mitigation in freertos applications," in *Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands*, ser. SBCCI '17. New York, NY, USA: ACM, 2017, pp. 84–89. [Online]. Available: http://doi.acm.org/10.1145/3109984.3110008

[38] A. Martínez-Álvarez, F. Restrepo-Calle, S. Cuenca-Asensi, L. M. Reyneri, A. Lindoso, and L. Entrena, "A hardware-software approach for on-line soft error mitigation in interrupt-driven applications," *IEEE Trans. Dependable Sec. Comput.*, vol. 13, no. 4, pp. 502–508, 2016. [Online]. Available: https://doi.org/10.1109/TDSC.2014.2382593

[39] V. Fratin, D. Oliveira, C. Lunardi, F. dos Santos, G. Rodrigues, and P. Rech, "Code-dependent and architecture-dependent reliability behaviors," 06 2018, pp. 13–26.

[40] G. S. Rodrigues and F. L. Kastensmidt, "Soft error analysis at sequential and parallel applications in ARM cortex-a9 dual-core," in *2016 17th Latin-American Test Symposium (LATS)*. IEEE, Apr 2016. [Online]. Available: https://doi.org/10.1109/latw.2016.7483359

[41] F. Rosa, F. Kastensmidt, R. Reis, and L. Ost, "A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability," in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. IEEE, Oct 2015. [Online]. Available: https://doi.org/10.1109/dft.2015.7315164

[42] A. Chatzidimitriou, M. Kaliorakis, S. Tselonis, and D. Gizopoulos, "Performance-aware reliability assessment of heterogeneous chips," in *2017 IEEE 35th VLSI Test Symposium (VTS)*. IEEE, Apr 2017. [Online]. Available: https://doi.org/10.1109/vts.2017.7928940

[43] X. Iturbe, B. Venu, and E. Ozer, "Soft error vulnerability assessment of the real-time safety-related ARM cortex-r5 CPU," in *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, Sep 2016. [Online]. Available: https://doi.org/10.1109/dft.2016.7684076

[44] J. Blome, S. Mahlke, D. Bradley, and K. Flautner, "A microarchitectural analysis of soft error propagation in a production-level embedded microprocessor," in *In Proceedings of the First Workshop on Architecture Reliability*, 2005.

[45] 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, Sep 2011. [Online]. Available: https://doi.org/10.1109/tc.2010.60

[46] D. Ferraretto and G. Pravadelli, "Simulation-based fault injection with QEMU for speeding-up dependability analysis of embedded software," *Journal of Electronic Testing*, vol. 32, no. 1, pp. 43–57, Jan 2016. [Online]. Available: https://doi.org/10.1007/s10836-015-5555-z

[47] F. de Aguiar Geissler, F. L. Kastensmidt, and J. E. P. Souza, "Soft error injection methodology based on QEMU software platform," in *2014 15th Latin American Test Workshop - LATW*. IEEE, Mar 2014. [Online]. Available: https://doi.org/10.1109/latw.2014.6841910

[48] A. Holler, G. Macher, T. Rauter, J. Iber, and C. Kreiner, "A virtual fault injection framework for reliability-aware software development," in *2015 IEEE International Conference on Dependable Systems and Networks Workshops*. IEEE, Jun 2015. [Online]. Available: https://doi.org/10.1109/dsn-w.2015.16

[49] L. Wanner, S. Elmalaki, L. Lai, P. Gupta, and M. Srivastava, "VarEMU: An emulation testbed for variability-aware software," in *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODESISSS)*. IEEE, Sep 2013. [Online]. Available: https://doi.org/10.1109/codes-isss.2013.6659014

[50] M. Becker, D. Baldin, C. Kuznik, M. M. Joy, T. Xie, and W. Mueller, "Xemu: An efficient qemu based binary mutation testing framework for embedded software," in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT '12. New York, NY, USA: ACM, 2012, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/2380356.2380368

[51] R. Amarnath, S. N. Bhat, P. Munk, and E. Thaden, "A fault injection approach to evaluate soft-error dependability of system calls," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, Oct 2018. [Online]. Available: https://doi.org/10.1109/issrew.2018.00-28

[52] H. Schirmeier, M. Hoffmann, C. Dietrich, M. Lenz, D. Lohmann, and O. Spinczyk, "FAIL: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance," in *2015 11th European Dependable Computing Conference (EDCC)*. IEEE, Sep 2015. [Online]. Available: https://doi.org/10.1109/edcc.2015.28

[53] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "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, Jun 2014. [Online]. Available: https://doi.org/10.1109/dsn.2014.2

[54] G. Yalcin, O. S. Unsal, A. Cristal, and M. Valero, "FIMSIM: A fault injection infrastructure for microarchitectural simulators," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, Oct 2011. [Online]. Available: https://doi.org/10.1109/iccd.2011.6081435

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

[56] N. Foutris, D. Gizopoulos, J. Kalamatianos, and V. Sridharan, "Assessing the impact of hard faults in performance components of modern microprocessors," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, Oct 2013. [Online]. Available: https://doi.org/10.1109/iccd.2013.6657044

[57] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Analysis and characterization of ultra low power branch predictors," in *2018 IEEE International Conference on Computer Design (ICCD)*. IEEE, Oct 2018.

[58] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez, "Hamartia: A fast and accurate error injection framework," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, Jun 2018. [Online]. Available: https://doi.org/10.1109/dsn-w.2018.00046

[59] R. B. Tonetto, G. L. Nazar, and A. C. S. Beck, "Precise evaluation of the fault sensitivity of OoO superscalar processors," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar 2018. [Online]. Available: https://doi.org/10.23919/date.2018.8342082

[60] E. Cheng, P. Bose, S. Mitra, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, and J. A. Abraham, "Clear," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*. ACM Press, 2016. [Online]. Available: https://doi.org/10.1145/2897937.2897996

[61] V. B. Kleeberger, C. Gimmler-Dumont, C. Weis, A. Herkersdorf, D. Mueller-Gritschneder, S. R. Nassif, U. Schlichtmann, and N. Wehn, "A cross-layer technology-based study of how memory errors impact system resilience," *IEEE Micro*, vol. 33, no. 4, pp. 46–55, Jul 2013. [Online]. Available: https://doi.org/10.1109/mm.2013.67

[62] A. Vallero, A. Savino, G. Politano, S. D. Carlo, A. Chatzidimitriou, S. Tselonis, M. Kaliorakis, D. Gizopoulos, M. Riera, R. Canal, A. Gonzalez, M. Kooli, A. Bosio, and G. D. Natale, "Cross-layer system reliability assessment framework for hardware faults," in *2016 IEEE International Test Conference (ITC)*. IEEE, Nov 2016. [Online]. Available: https://doi.org/10.1109/test.2016.7805863

[63] A. Vallero, A. Savino, A. Chatzidimitriou, M. Kaliorakis, M. Kooli, M. R. Villanueva, G. D. Natale, A. Bosio, R. Canal, D. Gizopoulos, and S. D. Carlo, "SyRA: Early system reliability analysis for cross-layer soft errors resilience in memory arrays of microprocessor systems," *IEEE Transactions on Computers*, pp. 1–1, 2018. [Online]. Available: https://doi.org/10.1109/tc.2018.2887225

[64] 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*, pp. 1–15, 2018.

[65] 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*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 28:1–28:12.

[66] 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 Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[67] X. Inc. (2018) Zynq-7000 soc data sheet: Overview. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[68] ——. (2018) Linux repository from xilinx. [Online]. Available: https://github.com/Xilinx/linux-xlnx/tree/xlnx_3.14

[69] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[70] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.

[71] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of error in full-system simulation," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar 2014. [Online]. Available: https://doi.org/10.1109/ispass.2014.6844457

[72] J. Baggio, V. Ferlet-Cavrois, H. Duarte, and O. Flament, "Analysis of proton/neutron SEU sensitivity of commercial SRAMs-application to the terrestrial environment test method," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3420–3426, Dec. 2004. [Online]. Available: https://doi.org/10.1109/tns.2004.839135