IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 11, 2018, accepted July 18, 2018, date of publication July 23, 2018, date of current version August 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2858773

Impact of Microarchitectural Differences of RISC-V Processor Cores on Soft Error Effects

HYUNGMIN CHO
Department of Computer Engineering, Hongik University, Seoul 04066, South Korea
(e-mail: hcho@hongik.ac.kr)

This work was supported in part by the Hongik University New Faculty Research Support Fund and in part by the National Research Foundation of Korea (NRF) Grant through the Korea Government [Ministry of Science, ICT & Future Planning (MSIP)] under Grant 2017017414.

ABSTRACT In this paper, we compare how radiation-induced soft errors affect the execution results of user-level applications on different processor cores that implement the same instruction set architecture (ISA). We target two processor cores that support the same RISC-V ISA but have significant differences in their microarchitectural implementations (in-order versus out-of-order). The observed results from fault injection experiments show very strong correlations between the resulting effects from those two processor cores. This strong correlation property is not observed between the processor cores that have different ISAs. Based on this observation, we discuss how the resulting effects of soft errors on a target processor core can be predicted using a reduced set of fault injection experiments with a small number of benchmark applications. Using our heuristic method of selecting the applications to be used for the fault injection experiments on the target processor core, we achieved a high prediction accuracy with prediction errors of less than 7%. Our approach can be used for rapid error resilience evaluation of system designs that have the same ISA.

INDEX TERMS Error resilience, microarchitecture, processor core, soft error.

I. INTRODUCTION

Transient errors that cause bit-flips (*soft errors*) are one of the major reliability threats in digital systems [1]–[3]. Bit-flips in on-chip memory or sequential elements (latches and flip-flops) can result in fatal consequences such as undetected data corruptions and expensive system downtimes. Soft errors are usually caused by error sources that are difficult to predict or estimate the magnitude, such as cosmic ray or alpha particle strikes. Therefore, it is challenging to provide cost-effective soft error resilience.

Over the transistor technology generations, the device-level soft error rate (i.e., the probability of having a bit-flip per memory cell or flip-flop) has stayed at roughly the same level or even decreased. However, as technology scales, the system-level soft error rate increases because more components are integrated into a chip [3]–[6]. Therefore, it is important to study the impact of soft errors at the system-level and provide appropriate error protection techniques to meet the desired level of soft error resilience.

In this study, we focus on soft errors in flip-flops (*flip-flop soft errors*) because providing adequate error protection mechanisms for flip-flops generally involves a high overhead. For on-chip memory arrays such as a data cache, coding techniques for error detection or correction are commonly used [7], [8]. Combinational logic circuits are significantly less susceptible to soft errors [3].

Not every flip-flop soft error affects the execution results of a user-level application. Furthermore, depending on the application and the type of the resulting erroneous outcome, the system may require different error protection mechanisms. Therefore, accurately characterizing how soft errors in hardware components affect the execution results of user-level applications (*soft error effects*) is crucial in order to provide cost-effective soft error resilience [9], [10]. Many researchers have characterized such soft error effects on a wide range of designs, including processor cores, memory subsystems, on-chip interconnects, and accelerator components [11]–[16].

Studying the effects of soft errors on a given system can be a demanding process even for a single target application. The characterization of soft error effects is conducted by collecting statistical observations on how frequently the low-level soft errors result in erroneous execution results, such as undetected data corruptions [17]. Radiation testing can be used to characterize the effect of soft errors on physical silicon chips [18], [19]. During the design phase, the effects of soft errors can be modeled on a simulation or emulation environment using fault injection experiments [2], [20]. Owing to the fact that the possible flip-flop soft error space is extremely large (i.e., number of processor clock cycles × number of flip-flops), soft error characterization is performed using the Monte Carlo method which randomly selects the fault injection target. Consequently, we must collect a large number of fault injection samples to obtain statistically significant results. Moreover, even on the same system, the resulting effects can be drastically different for different applications (*application dependence*). For example, the probability that a soft error eventually results in undetected corruptions in application output, also known as silent data corruption (SDC), ranges from 0.02% to 1.78% on the LEON3 processor core depending on the application [11].

Unfortunately, this soft error effect characterization may not be a one-time process. When designing a system, multiple types of designs can be considered to meet various design requirements, such as performance, area, and power consumption. Traditionally, only a few system designs were considered as candidates for a given environment and application. For instance, many embedded systems adopt simple in-order processor cores owing to area and power limitations, whereas high-performance computing systems use complex out-of-order processor cores to maximize single thread performance. However, as the computing environment and application diversifies, an application may run on multiple different designs. If multiple processor cores support the same instruction set architecture (ISA), the same application binary can be executed on the processor cores that have different implementations (i.e., microarchitectural differences). For example, the big.LITTLE architecture of ARM combines energy-efficient in-order processor cores as "LITTLE" cores and high-performance out-of-order processor cores as "big" cores [21]. Depending on the operational mode, a single instance of an application may migrate between the cores to adapt to the execution performance and the overhead. Therefore, to design and evaluate error resilient systems, we must study the soft error characteristics on a wide variety of environments.

In this study, we compare the resulting effects of soft errors on two processor cores with different microarchitecture implementations of the same ISA. Our fault injection results show that even with the profound differences (in-order versus out-of-order) of the two processor core designs, the application dependence aspects of the soft error effects are strongly preserved. In other words, given that we have characterized the soft error effects on a processor core, we can predict the soft error effects on another target processor core of the same ISA without fully performing fault injection experiments on the target system. This strong correlation is only observed between the processor cores of the same ISA, not across different ISAs. This observation helps to evaluate the error resilience of new system designs by reducing the burden of time and resources required to perform fault injection experiments for characterizing the effects of soft errors.

Some soft error studies have assumed that the application dependence can be separated from the soft error characteristics of the underlying microarchitecture design without a quantified comparison by using fault injection experiments [22]. In this study, we report fault injection and comparison results to substantiate how microarchitectural differences affect the characteristics of soft error effects.

We also present a heuristic method to select a small set of applications to be used for the soft error effects prediction on a target system. Guided by this heuristic method, a reduced set of fault injection experiments with only three applications on a target processor core models the soft error effects of the rest of the applications with less than 7% difference from the actual fault injection results.

The rest of this paper is organized as follows: Section II introduces the RISC-V ISA, which is the common ISA of the two processor cores that are compared in this study. Section III explains the erroneous outcome types of user-level applications affected by soft errors. Section IV presents our setup for fault injection experiments. Section V compares the resulting fault injection results from the processor cores. In Section VI, we demonstrate our method for predicting soft error effects on a target processor core. Finally, Section VII introduces the related work, and Section VIII presents the conclusion of the paper.

II. RICV-V ISA

To compare the soft error effects on different processor cores of the same ISA, we select RISC-V as the target ISA [23]. The RISC-V ISA is a royalty-free open ISA that is adopted across academia and various industries. A variety of microprocessor designs are actively being developed to support the RISC-V ISA for various environments [24]. For example, ORCA specifically targets Field-Programmable Gate Array (FPGA) environments, and RV12 focuses on highly-configurable embedded designs. Given that many of the RISC-V processor cores are available as open source, RISC-V is an ideal target for comparing different processor core implementations of the same ISA.

Among the available RISC-V processor cores, we chose Rocket [25] and the Berkeley Out-of-Order Machine (BOOM) [26] for the main comparison in this study. We also include the PULPino[1] RISC-V core, which is a 32-bit RISC-V core developed at ETH Zürich, for an extended comparison. The majority of the currently available RISC-V processor cores implement a simple in-order processor pipeline for low-cost energy-efficient designs. We use Rocket to represent this class of in-order processor cores. Rocket is a core component of the Rocket Chip Generator, which creates RISC-V-based system designs in various configurations. BOOM represents the other extreme of the microarchitecture design spectrum. BOOM is a superscalar out-of-order execution microprocessor that targets a high-performance processor design. It is one of the few open-source implementations

[1] https://www.pulp-platform.org/

of a fully functional out-of-order microprocessor. BOOM supports flexible configurations for diverse microarchitectural design parameters, such as instruction issue width and register file size. Depending on the configuration, the achievable performance of BOOM is comparable to those of industrial processor core implementations. Configuration parameters for BOOM that are used in our fault injection experiments are listed in Table 1. Tables 2 and 3 list the number of flip-flops (i.e., targets of flip-flop fault injections) in Rocket and BOOM, receptively.[2]

TABLE 1. BOOM processor core configuraion parameters.

Configuration parameter	Value
Reorder buffer entry	16 instructions
Issue width	2 instructions per cycle
Issue slot entries	6 instructions
Number of physical registers	56 registers
Speculated branch count	4 branches
Branch predictor	gshare
Global branch history length	8 branches

TABLE 2. Number of flip-flops in submodules of Rocket.

Submodule	Number of flip-flops
Register file	1,984
Control and status registers (CSR)	984
Instruction buffer	76
Integer pipeline	872
Multiplier/Divider	214
Total	4,130

TABLE 3. Number of flip-flops in submodules of BOOM.

Submodule	Number of flip-flops
Register file	3,964
CSR	1,238
Instruction fetch	854
Register rename	1,547
Instruction issue	650
Load/Store unit	1,941
Arithmetic/logic unit	1,119
Reorder buffer	1,707
Branch prediction	465
Total	13,485

III. SOFT ERROR EFFECTS

We distinguish the resulting effects of soft errors in the following categories, similar to the existing fault injection studies for soft errors [27], [28]:

1) **Silent Data Corruption (SDC):** The application completes its execution normally without any indication of errors. However, upon the completion of the application, the results obtained from the execution differ from those of an error-free (golden) execution. For our benchmark applications, we verify whether the obtained output files and the standard output (i.e., `stdout` console output) match the golden output.
2) **Unexpected Termination (UT):** The application terminates abnormally with an error indication, such as a memory access violation, kernel panic, and arithmetic exception.
3) **Hang:** The application does not produce any result or terminates within a specified timeout limit, which is set to 2× the nominal execution time.
4) **Vanished:** The obtained program output matches that of the error-free execution. The injected error may have affected the architectural states such as registers or program variables, but the error has been masked or overwritten during the execution.

Because these outcome types are mutually exclusive, the outcome type of each application execution with a fault injection (*fault injection run*) is classified into one of these outcome types. We denote the observed rate of each outcome type as p(outcome type, application, processor). For example, p(SDC, *crafty*, Rocket) is the observed SDC rate from Rocket when running the *crafty* application.

IV. EXPERIMENT SETUP

To accurately model the effects of flip-flop soft errors, the fault injection experiment should use an injection technique that can correctly model how a soft error injected into a flip-flop behaves in the system. Fault injection techniques which model the soft errors by injecting bit-flips into the high-level abstracted states such as architectural registers or program variables do not accurately quantify the effects of flip-flop soft errors [11]. Moreover, because we compare the differences according to the microarchitecture implementations in this study, we perform fault injection experiments on a platform that can model the low-level implementation details of given processor designs. Fault injection techniques based on instruction-level simulators (e.g., [10], [29], [30]) cannot be used for this study because the differences in the microarchitecture-level implementations are not modeled in those platforms. RTL simulations are not suitable for collecting a large number of fault injection samples owing to their slow simulation throughput.

We use an FPGA emulation of the processor cores to perform fault injection experiments while executing user-level applications. We use the Xilinx Zynq-7000 FPGA device to emulate the processor cores along with a flip-flop-level fault injector module. Studying the effects of soft errors requires a large number of fault injection samples. We use multiple FPGA boards to collect fault injection samples from diverse benchmark applications. Each FPGA board

[2] Owing to the FPGA capacity limitations, we did not enable the floating-point unit for both processor cores.

FIGURE 1. Xilinx Zynq-7000 FPGA boards.

includes an instance of the target system, and we run multiple fault injection instances simultaneously. Fig. 1 shows the array of FPGA boards used for collecting fault injection results.

A. SOFT ERROR MODEL

We characterize the effects of soft errors by observing how frequently we obtain an erroneous outcome (i.e., SDC, UT, or Hang) when we inject a single bit-flip in a randomly selected flip-flop during the application execution. The injected bit-flip represents a soft error in a flip-flop, and it may eventually result in an application-level erroneous outcome. Systems with a different number of flip-flops would have a different number of bit-flips per period. Therefore, to evaluate the error resilience level of a given system, the soft error rate of the underlying device technology and the number of flip-flops in the system have to be considered for the quantification.

In this study, however, we focus on comparing the characteristics related to the application dependence rather than comparing the raw error rates of two different designs. Therefore, we inject a single bit-flip for each fault injection run for both processor cores even though Rocket and BOOM have different numbers of flip-flops.

For each application, we collect more than 80,000 fault injection samples from each processor core (for a total 1.6 million fault injection samples). If we assume a normal approximation of the binomial distribution, the resulting accuracy is better than ±0.1% with 95% confidence when the observed rate is 1%.

B. SOFT ERROR FAULT INJECTION ON FPGA

We use the FPGA emulation setup of the Rocket Chip Generator[3] as our base platform for fault injection, which achieves a 50MHz clock speed for Rocket and BOOM. In this setup, a host CPU (ARM Cortex-A9) on the processing system (PS) side manages the RISC-V processor cores programmed on the programmable logic (PL) side of the FPGA (Fig. 2). The host CPU in PS can trigger the execution of a single executable binary or the Linux operating system on the target processor core in PL. In this work, we execute a single binary executable without an operating system (i.e., bare-metal program) on the target processor core for each fault injection run. After the target processor core finishes the execution, the host CPU collects the execution results and classifies the outcome type.

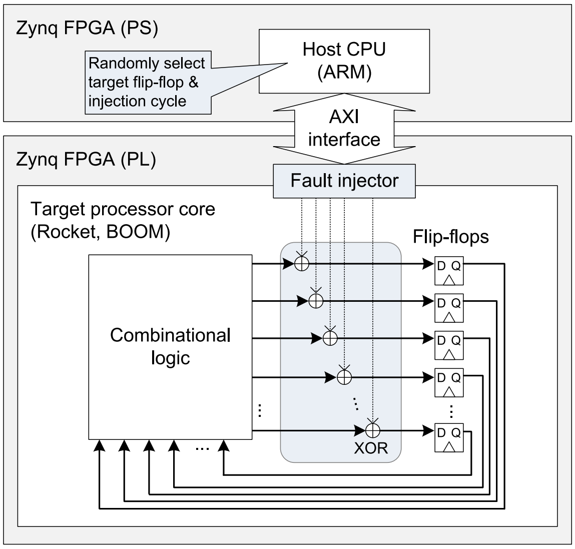


FIGURE 2. Target processor core emulation on the Xilinx Zynq FPGA platform with a flip-flop fault injector.

Rocket and BOOM are created using the Chisel hardware construction language [31]. When creating the emulation environment for the processor cores on FPGA, the Chisel compiler produces a Verilog file that encodes the netlist information of the design. We use this netlist Verilog file to insert a flip-flop-level fault injector automatically. Fig. 3 describes the flow of generating the flip-flop fault injector module on FPGA. We synthesize the given netlist file using Synopsys Design Compiler to obtain a list of flip-flops in the target processor core design. For every flip-flop in the list, we add an extra XOR gate in the D input to connect an fault injection signal. The added fault injection signals and the XOR gates are also depicted in Fig. 2. The fault injection signals are controlled by the host CPU through an AXI interface that connects the PS and the PL side. During each fault injection run, the host CPU randomly selects a target flip-flop for fault

[3] https://github.com/ucb-bar/fpga-zynq

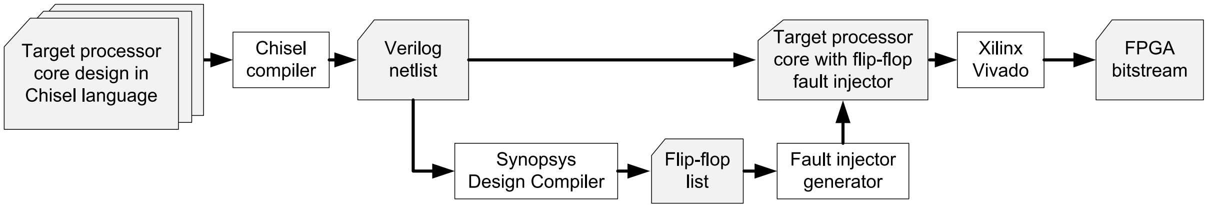


FIGURE 3. An automated flow to inject a flip-flop fault injector on FPGA.

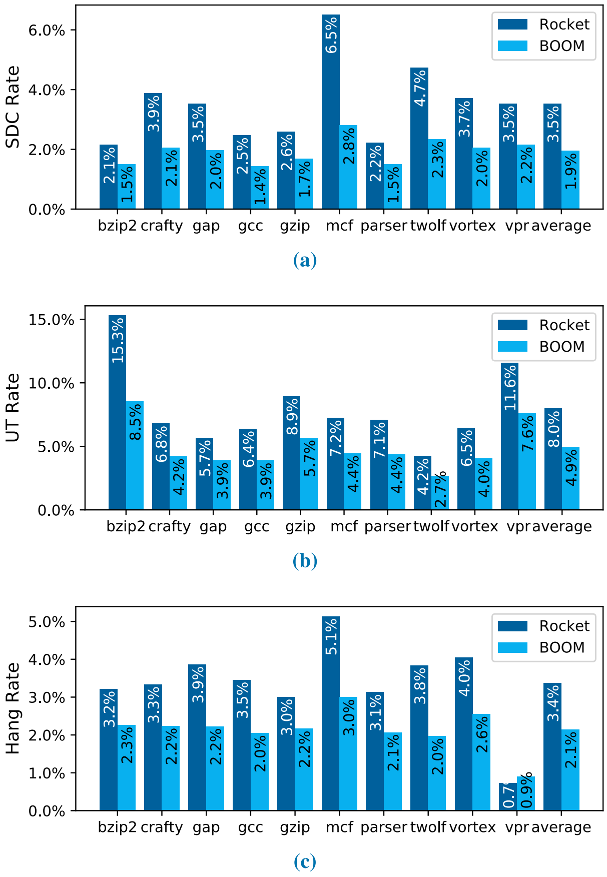
injection and asserts the corresponding fault injection signal as high at a randomly selected fault injection cycle. The value of the flip-flop is inverted to model a bit-flip caused by a soft error.

C. BENCHMARK APPLICATIONS

We use the SPECINT 2000 applications and the MinneSPEC input datasets [32], which is a benchmark suite widely used for fault injection studies [33]–[35]. *perlbmk* and *eon* are excluded from the fault injection experiments because they require floating-point instructions or extensive file system support that is not modeled in our fault injection system.

V. FAULT INJECTION RESULTS

Fig. 4 presents the observed erroneous outcome rates from Rocket and BOOM across the benchmark applications and the arithmetic mean of the outcome rates. In general, the erroneous outcome rates of BOOM are lower than that of Rocket. The average SDC rate of BOOM is 1.9%, whereas Rocket has an average SDC rate of 3.5% (Fig. 4a). Although BOOM is a more complex out-of-order execution microprocessor, not every flip-flop in the processor core has the same impact on the application execution. For example, faults occurring in the branch prediction unit may alter the microarchitectural states during the execution. However, mistakenly executed branches will be resolved before the instructions retire from the processor core and alter the application results. In addition, BOOM has more physical registers than Rocket, which does not have a register renaming technique. This means that the chance of having a bit-flip on a register that can affect the application outcome is lower in BOOM as we inject only one bit-flip per fault injection run.



(a)

(b)

(c)

FIGURE 4. Observed erroneous outcome rates of Rocket and BOOM. (a) SDC rates. (b) UT rates. (c) Hang rates.

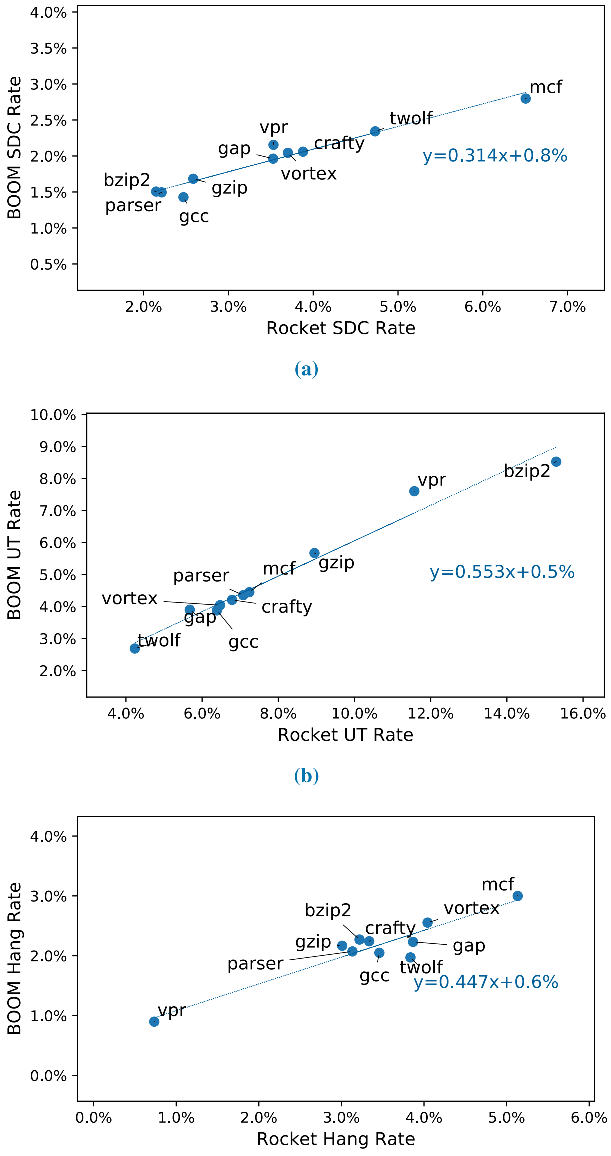
TABLE 4. Observed erroneous outcome rates of each submodule in Rocket.

Submodule	SDC	UT	Hang
Register file	3.32%	12.17%	1.04%
CSR	5.88%	4.30%	8.17%
Instruction buffer	0.50%	1.08%	0.28%
Integer pipeline	2.40%	4.93%	4.36%
Multiplier/Divider	0.17%	0.39%	0.05%

As we discussed earlier in Section IV-A, however, having lower erroneous outcome rates per injected fault does not mean that BOOM has higher error resilience than Rocket. Because BOOM has more flip-flops than Rocket, it has a higher chance of having a flip-flop soft error during application execution. Tables 4 and 5 show the breakdown of error rates from each submodule in Rocket and BOOM. For example, the register file modules in both processor cores have similar contributions for three erroneous outcome types. However, erroneous outcome rates resulting from the soft errors in BOOM's register file are lower as BOOM has a larger number of physical registers. Also, some components in BOOM (e.g., branch predictor) have very low contributions to erroneous outcomes. The focus of this study is to analyze whether the differences in the erroneous outcome rates depending on the application are preserved across the two

TABLE 5. **Observed erroneous outcome rates of each submodule in BOOM.**

Submodule	SDC	UT	Hang
Register file	1.89%	8.71%	0.96%
CSR	3.91%	5.43%	7.68%
Instruction fetch	2.57%	7.32%	0.99%
Register rename	2.37%	5.68%	3.34%
Instruction issue	2.35%	0.85%	3.11%
Load/Store unit	1.45%	3.67%	2.40%
Arithmetic/logic unit	1.16%	0.68%	0.17%
Reorder buffer	1.16%	0.78%	1.49%
Branch prediction	0.80%	0.05%	0.37%



(a)

(b)

(c)

FIGURE 5. **Observed erroneous outcome rates comparison between Rocket and BOOM. (a) SDC rates. (b) UT rates. (c) Hang rates.**

processor cores rather than comparing the scale of the raw error rates from two different implementations.

The observed outcome rates indicate very strong correlations between the soft error effects of the two processor cores. For example, among the evaluated benchmark applications, *mcf* shows the highest SDC rate for Rocket and also for BOOM. The SDC rate of *twolf* is the second highest on both processor cores. Similar trends were observed across other applications. The UT and Hang outcome rates also suggest similarities in the results between the two processor cores. That is, the "*rank*" of the applications according to the outcome rates, i.e., the ordering of the soft error vulnerabilities, does not have a noticeable difference between the two processor cores. To predict the soft error effects of a processor core using the fault injection results from another processor core, we need a stronger correlation in the results than just mere similarities in the soft error vulnerability ranking.

A. CORRELATION COMPARISON

To compare and visualize the correlation, Fig. 5 plots the observed outcome rates from the two processor cores in a two-dimensional space, where the X and Y axes represent the outcome rates from Rocket and BOOM, respectively. Each plot also shows a linear trend line (i.e., $y = Ax + B$) that minimizes the following squared error:

$$\sum_{a_i \in applications} \left\{p(T, a_i, BOOM) - A \cdot p(T, a_i, Rocket) + B\right\}^2 \tag{1}$$

where T is the evaluated erroneous outcome type, one of SDC, UT, or Hang.

As the plots show, the observed erroneous outcome rates fit the linear trends very well for all three types of outcomes. This means that we can accurately predict the outcome rates of a target processor core using the established soft error effect characteristics of an existing processor core.

To quantify the level of correlations, we calculated the Pearson correlation coefficients. Table 6 shows the correlation coefficients between the observed outcome rates from Rocket and BOOM. For all three outcome types, the Pearson correlation coefficient is larger than 0.94, which is a very strong positive correlation.

In contrast, such strong correlation is not observed between the processor cores that have different ISAs. Table 6 also compares the correlations in erroneous outcome rates between the RISC-V processor cores and other two processor cores, namely, LEON3 and IVM, which use different ISAs. LEON3 [36] is an in-order single-issue processor core that is similar to Rocket, and IVM [27] is an out-of-order superscalar processor core that is similar to BOOM. LEON3 uses the SPARC ISA, and IVM uses the ALPHA ISA. We calculated the correlation coefficients using the published fault injection results for LEON3 and IVM in our previous

TABLE 6. Pearson correlation coefficients between the observed outcome rates from the processor cores. Correlation coefficients between Rocket and BOOM are highlighted in bold.

ISA		RISC-V	SPARC	ALPHA
	Processor	BOOM	LEON3	IVM
RISC-V	Rocket	**SDC: 0.973** **UT: 0.986** **Hang: 0.942**	SDC: 0.722 UT: 0.814 Hang: 0.027	SDC: 0.776 UT: 0.670 Hang: 0.027
	BOOM	–	SDC: 0.754 UT: 0.723 Hang: -0.043	SDC: 0.714 UT: 0.690 Hang: 0.112
SPARC	LEON3	–	–	SDC: 0.477 UT: 0.558 Hang: -0.020

work [11], which uses the same SPECINT2000 benchmark applications and input datasets as this work. Unlike the correlations between the RISC-V processor cores, the correlations across different ISAs are not apparent. Some cases even show negative correlations (e.g., the Hang rates between LEON3 and BOOM). The weak correlations mean that accurately predicting erroneous outcome rates is difficult across different ISAs without conducting thorough fault injection experiments on the target processor core.

B. CORRELATION COMPARISON UNDER MULTIPLE BIT-FLIPS

As technology scales, multiple bit-flips per single event (single-event multiple upsets or SEMUs), are becoming more critical. To verify if the correlation trends in the soft error effects across different processor cores still exist under SEMU cases, we extended our fault injection experiments to inject multiple flip-flop bit-flips per fault injection run. Modeling SEMUs is not straightforward compared to single-upset cases because it may involve more variables such as the number of flipped bits per event and the locations of the upsets. In this work, we studied the following two versions of SEMU cases to assess if the correlation can be expected on other SEMU models.

1) **Random pair**: On each fault injection run, we inject bit-flips in two flip-flops that are randomly selected regardless of their location.
2) **Nearest neighbor**: On each fault injection run, we randomly select a flip-flop and the selected flip-flop's nearest neighbor as the targets of bit-flip fault injections. The nearest neighbor is identified based on the placement result of the target processor core using Synopsys ICC.

As Table 7 shows, strong correlations are observed under both SEMU models as well, meaning that our observations are not limited to single event upsets only. Because the characteristics of SEMUs are highly related to the technology node of the target system, future work that studies on specific technology nodes should address the correlations using the corresponding SEMU models.

TABLE 7. Pearson correlation coefficients between Rocket and BOOM under SEMU error models.

Random pair		Nearest neighbor	
SDC:	0.979	SDC:	0.993
UT:	0.937	UT:	0.958
Hang:	0.936	Hang:	0.943

C. CORRELATION COMPARISON WITH THE PULPINO RISC-V PROCESSOR CORE

We include another RISC-V processor core for comparison to verify whether the observed correlation trends can be extended to a broader range of RISC-V processor cores. We extended our fault injection experiments to the PULPino processor core. Unlike Rocket and BOOM, PULPino is created using System Verilog HDL. We selected the following five benchmark programs for comparison between Rocket and PULPino: Convolution, FFT, Motion detection, Neural network, and Sudoku solver. This is because the application execution environment on FPGA for PULPino has limited support for system calls, without file I/Os that are required to run SPEC benchmark applications. Also, the comparison is limited to SDC rate comparison because the exception handlers are implemented differently and result in different UT or Hang behaviors. Please note that this discrepancy is not applied to the situation where someone compares different processor candidates on the same software environment. BOOM is not included in this comparison since BOOM does not support RV32, the 32-bit version of RISC-V ISA that PULPino implements. The observed SDC rates between Rocket and PULPino (Fig. 6) also show a strong correlation (correlation coefficients higher than 0.97), which means the correlations in soft error effects on RISC-V processors exist on a broad range of RISC-V processor core implementations.

VI. SOFT ERROR EFFECT PREDICTION

Given that the observed erroneous outcome rates generally follow a linear relationship between the processor cores of the same ISA, we need to collect the fault injection results using

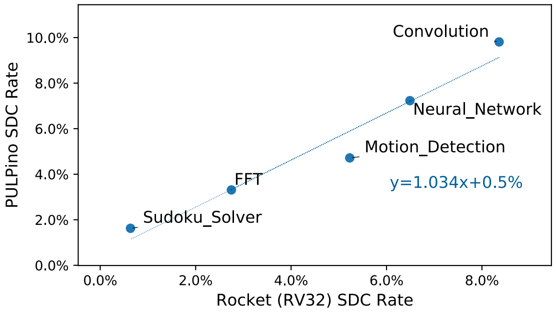


FIGURE 6. Observed SDC rates comparison between Rocket and PULPino.

only a few applications on the target processor core to predict the outcome rates of the remaining applications. Using the observed fault injection results from the profiling runs on the target processor core, we can obtain a linear relationship with the established fault injection results of an existing processor core of the same ISA. Then, for the other applications that have not been characterized on the target processor core, we can predict the soft error effects using the obtained linear relationship.

The trend lines in Fig. 5 have non-zero bias values (i.e., B is not zero in (1)). For example, the trend line for the UT rates in Fig. 5b has the B value of 0.5%. This represents the fact that between the processor cores, there are certain levels of inherent differences in erroneous outcome rates regardless of the application. Consequently, we need to obtain fault injection results using at least two applications on the target processor core to project the linear relationship with the existing soft error characterization. Beyond the two applications, increasing the number of benchmark applications for profiling runs on the target processor core may increase the accuracy of the projection, but at the same time, it increases the time and resource required by the fault injection runs since we have to collect a larger number of fault injection samples.

A. MEASURING PREDICTION ACCURACY

We measure the required number of applications to perform profiling runs on the target processor core to meet the desired level of prediction accuracy. We divide the benchmark applications into the following two groups:

1) **Training group**: For each application in this group, we use the observed outcome rates obtained from the fault injection runs on the target processor core. Therefore, it is desirable to obtain high accuracy in the outcome rate prediction with a small number of applications in the training group.
2) **Test group**: We evaluate the accuracy level of the soft error effects prediction using the applications in this group.

Using the observed erroneous outcome rates of the training group applications, we derive the linear projection ($y = \hat{A}x + \hat{B}$) that minimizes the following squared error for each outcome type T.

$$\sum_{a_i \in training\ group} \left\{p(T, a_i, P_{target}) - \hat{A} \cdot p(T, a_i, P_{existing}) + \hat{B}\right\}^2 \tag{2}$$

where $P_{existing}$ is the processor core for which we already have established soft error characterizations, and P_{target} is the target processor core on which we run profiling fault injection runs using the training group applications. The outcome rate prediction ($\hat{p}$) on the target processor core for each test group application (a_{test}) is calculated using the following equation:

$$\hat{p}(T, a_{test}, P_{target}) = \hat{A} \cdot p(T, a_{test}, P_{existing}) + \hat{B} \tag{3}$$

We evaluate the prediction error (E) for the outcome rates using the relative difference between the prediction and the actual observation on the target processor core.

$$E(T, a_{test}) = \frac{\left|\hat{p}(T, a_{test}, P_{target}) - p(T, a_{test}, P_{target})\right|}{p(T, a_{test}, P_{target})} \tag{4}$$

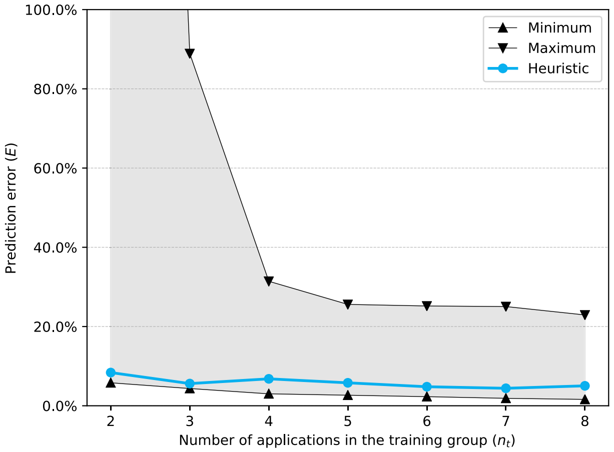


FIGURE 7. Prediction error comparison for training group application choices.

The accuracy of the outcome rate prediction is highly dependent on the choice of the applications to be included in the training group. The number of possible choices is $\binom{n}{n_t}$, where n is the number of benchmark applications and n_t is the number of applications in the training group. Fig. 7 shows the resulting accuracy range of all possible combinations of the training group selections. For simplicity, we report the average of E across the test group applications and all three types of erroneous outcomes to represent the obtained accuracy level in Fig. 7. It is evident that an arbitrary choice of the applications for the training group may result in a poor prediction accuracy. For example, when $n_t = 3$, a bad selection of the training group applications can result in a prediction error higher than 80%. Determining the training group that achieves the minimum prediction error (i.e., the highest accuracy) is not possible before performing fault injection experiments on the target processor core.

B. HEURISTIC APPROACH TO SELECT APPLICATIONS FOR FAULT INJECTION

Instead, we propose a heuristic method to choose applications for the training group. The main strategy is to select a group of applications that have a wide spectrum of outcome rates on $P_{existing}$. The heuristic method works in the following way when $n_t \geq 2$:

1) For a given outcome type T, sort the applications in an increasing order of the outcome rates on $P_{existing}$.
2) Select the two applications that have the lowest and the highest outcome rates and add those to the training group.
3) We select the rest of the applications (i.e., $n_t - 2$ applications) whose outcome rates are most "evenly" distributed between those of the previously selected two applications. This is achieved when the following quantity is minimized:

$$\sum_{i=1}^{n_t-1} \left\{ p(T, a_{i+1}^T, P_{existing}) - p(T, a_i^T, P_{existing}) \right\}^2 \quad (5)$$

where a_i^T is the application whose the outcome rate of T on $P_{existing}$ is the i^{th}-highest in the current training group. (5) sums the square of the intervals between the outcome rates of the applications in the group, which is minimized when the outcome rates of the selected training group applications are evenly distributed between the minimum and maximum outcome rates.

The goal of the heuristic approach is to select a minimal number of applications to perform fault injection study on a new platform. Although the soft error effects across different processors show highly correlated results that are very close to a linear relationship, the observed outcome rates may have some deviations due to the sampling errors caused by Monte Carlo method that selects a certain number of random fault injection samples. The obtained linear projection should capture the overall correlation trend despite such noisy observations. The heuristic approach tries to achieve this by selecting a group of applications that show a broad spectrum of erroneous outcome rates. Because the selected applications show diverse outcome rates including the maximum and minimum observed outcome rates, small sampling errors do not affect the derived linear projection (i.e., $\hat{A}$ and $\hat{B}$ in (3)) on a large scale.

The resulting accuracy level when we select the training group application using the heuristic method is also plotted in Fig. 7. The training group decided by this heuristic method results in prediction errors that are very close to the best (i.e., the minimum prediction error) case. From $n_t \geq 3$, the average of the prediction errors is better than 7%.

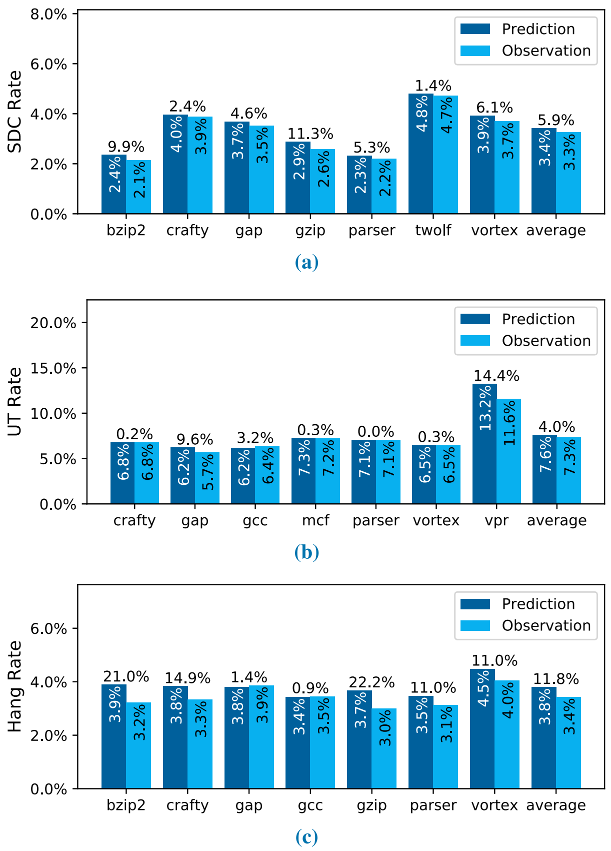
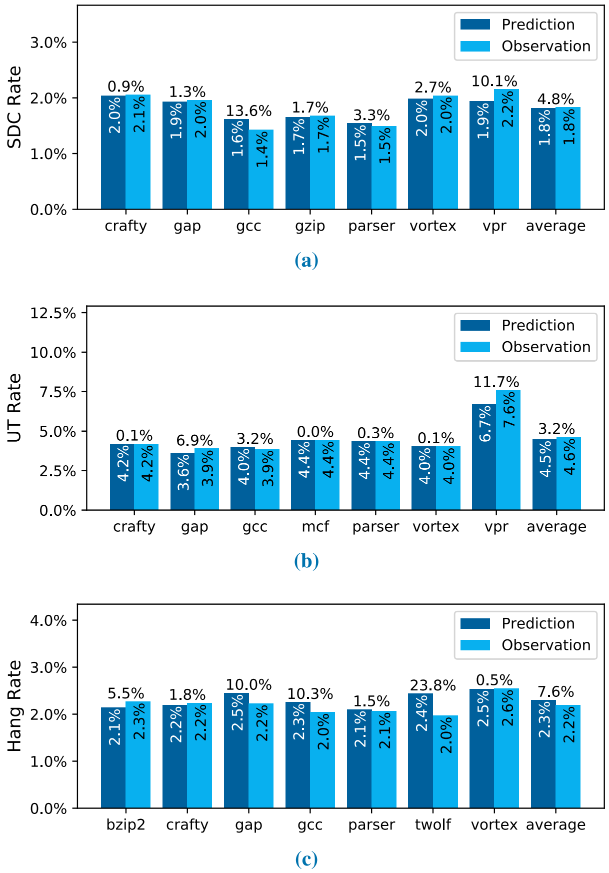


FIGURE 8. Accuracy comparison of the estimated outcome rate prediction and observed results from fault injection experiments for Rocket. (a) SDC rates. (b) UT rates. (c) Hang rates.

As a detailed example of the outcome rate prediction, Figs. 8 and 9 compare the predicted outcome rates and observation results from fault injection runs on Rocket and BOOM, respectively. Fig. 8 shows predictions for Rocket using the fault injection results of BOOM, and the predictions in Fig. 9 use the fault injection results of Rocket to predict the outcome rates of BOOM. The numbers on the top of the bars in Figs. 8 and 9 indicate the prediction error (E) values.[4] For this example, n_t is set to 3, meaning that we assume the profiling on the target processor core is performed using three applications in the training group. For instance, to estimate the SDC rates for BOOM in Fig. 9a, the training group consists of *bzip2*, *mcf*, and *twolf*. The graph shows the prediction accuracy levels for the rest of the applications in the test group. On average, the resulting prediction error is only 4.8%.

The prediction accuracy for Hang rates in Figs. 8 and 9 is a little worse than other outcome types. The prediction accuracy may degrade if there are no pronounced differences between the applications. Because the primary strategy criteria of the heuristic method for selecting applications for fault injection is selecting applications that can address a wide range of outcome rates, the approach works better if the applications have higher diversity in the outcome rates. In contrast, the heuristic method may have difficulties for accurately predicting individual outcome rates if there are small differences among the applications because the derived

[4] The values on top of the *average* bars are the arithmetic means of the prediction errors across the test group applications.



(a)

(b)

(c)

FIGURE 9. **Accuracy comparison of the estimated outcome rate predictions and observed results from fault injection experiments for BOOM. (a) SDC rates. (b) UT rates. (c) Hang rates.**

linear projection could be biased by the minimum and the maximum outcome rates. In Figures 8 and 9, the standard deviation of the observed outcome rates is much lower for the Hang rates compared to other outcome types. For example, in the observed outcome rates of Rocket, the standard deviation of the UT rates is 1.80%, whereas the standard deviation is 0.36% for the Hang rates. Although the heuristic method may work less effectively on outcome types with smaller diversity in the outcome rates, a lower difference in the erroneous outcome rates means there is a smaller need for accurately predicting the individual application's outcome rates to assess the soft error resilience of the target system.

VII. RELATED WORK

Many researchers have proposed various ways of reducing the overhead of studying the effects of soft errors on a given system.

A. REDUCING THE NUMBER OF APPLICATIONS TO PERFORM FAULT INJECTION

Lu *et al.* [28] proposed models to predict the SDC rates of target applications without performing fault injection experiments on every benchmark application. These models are based on instruction-level fault injection profiling of training applications, which do not capture the microarchitectural implementation details. In addition, they focus on the SDC-type outcomes only, unlike our method which covers all types of erroneous outcomes. Bronovetsky *et al.* [37] uses a machine-learning approach to model the soft error effects, but the target applications are limited to linear algebra applications.

B. ANALYTICAL METHOD

A large body of literature exists that evaluates the Architectural Vulnerability Factor (AVF), which quantifies the soft error masking rate of a system, and estimates the error resilience of the system using the AVF value [22], [38]–[40]. These AVF-based approaches generally provide loose bounds compared to the predictions based on fault injection results.

C. HYBRID APPROACH

Mirkhani *et al.* [41] created a method for modeling soft error effects of complex designs using fast local simulations. The actual outcome rates of the target system are calculated from the collected local simulation results. Hari *et al.* [42], [43] proposed a method to predict the soft error effects using fault injections on selected program sites only.

These existing approaches are primarily focused on predicting the erroneous outcome rates of the target applications on the same system. Therefore, these approaches are orthogonal to our method which correlates the soft error effects across different target systems.

D. FAULT INJECTION TECHNIQUES

Software-level and architecture-level fault injection techniques model soft errors using bit-flips in abstracted states, such as program variables or software-visible architectural registers [29], [30], [44], [45]. These mechanisms provide fast fault injection runs, but the obtained injection results may contain inaccuracies [11]. Co-simulation approaches that combine two or more levels of abstraction can accelerate the performance of fault injection runs [27], [46]. Flip-flop soft errors are injected on low-level simulation or emulation environments, while the long-term effects of the errors that affect the execution of the application are modeled using fast high-level simulators, such as instruction-level simulators. These approaches provide accurate error modeling and accelerate the overall simulation runs. However, some flip-flop soft errors may not be abstracted into the high-level simulators, and those errors can be a source of inaccuracies in the results or require extended RTL simulation time. Flip-flop-level fault injections, which model flip-flop soft errors using RTL simulations, can model accurate soft error behaviors [2]. In our work, we overcome the slow execution speed of RTL simulations using FPGA platforms. When such FPGA-based fault injection platforms are not available, our mechanism for predicting soft error behaviors in the new target platform can reduce the fault injection efforts.

VIII. CONCLUSION

Characterizing the effects of soft errors on a system can be a major obstacle for the rapid development of error resilient systems. Studying soft error effects often involves time-consuming simulation or emulation runs to collect a large number of fault injection samples for a wide variety of target applications. In this study, we quantitatively compared the soft error effects on two processor cores that have different microarchitectural implementations of the same ISA. Although the raw values of the erroneous outcome rates from the two processor cores are different, the outcome rates depending on the applications have very strong correlations. Between the processor cores of the same ISA, the erroneous outcome rates of the applications can be approximated by a simple linear relationship.

Utilizing the correlation, we demonstrated that the erroneous outcome rates of a target processor core can be predicted using an existing soft error effect characterization established on a processor core of the same ISA. To minimize the profiling efforts required on the target processor core, we also created a heuristic method to determine the group of applications to collect fault injection samples on the target processor core. With only three applications in the training group, the erroneous outcome rates on the target processor core can be predicted with only 7% prediction error. This approach can greatly reduce the amount of time required for evaluating the error resilience of a given system and potentially assist in developing new soft error protection techniques.

REFERENCES

[1] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[2] P. N. Sanda *et al.*, "Soft-error resilience of the IBM POWER6 processor," *IBM J. Res. Develop.*, vol. 52, no. 3, pp. 275–284, May 2008.

[3] N. Seifert *et al.*, "Soft error susceptibilities of 22 nm tri-gate devices," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 6, pp. 2666–2673, Dec. 2012.

[4] N. Seifert, "Radiation-induced soft errors: A chip-level modeling perspective," *Found. Trends Electron. Des. Autom.*, vol. 4, nos. 2–3, pp. 99–221, Feb. 2010.

[5] N. Seifert *et al.*, "Soft error rate improvements in 14-nm technology featuring second-generation 3D tri-gate transistors," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, pp. 2570–2577, Dec. 2015.

[6] P. Nsengiyumva *et al.*, "A comparison of the SEU response of planar and FinFET D flip-flops at advanced technology nodes," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 1, pp. 266–272, Feb. 2016.

[7] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proc. MICRO*, Chicago, IL, USA, Dec. 2007, pp. 197–209.

[8] J. M. Hart *et al.*, "A 3.6 GHz 16-core SPARC SoC processor in 28 nm," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 19–31, Jan. 2014.

[9] E. Cheng *et al.*, "CLEAR: Cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores," in *Proc. DAC*, Austin, TX, USA, Jun. 2016, pp. 1–6.

[10] M. Shafique, S. Rehman, P. V. Aceituno, and J. Henkel, "Exploiting program-level masking and error propagation for constrained reliability optimization," in *Proc. DAC*, Austin, TX, USA, May 2013, p. 17.

[11] 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 *Proc. DAC*, Austin, TX, USA, May/Jun. 2013, pp. 1–10.

[12] H. Cho, E. Cheng, T. Shepherd, C.-Y. Cher, and S. Mitra, "System-level effects of soft errors in uncore components," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1497–1510, Sep. 2017.

[13] 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.

[14] 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 Trans. Comput.*, vol. 60, no. 9, pp. 1260–1273, Sep. 2011.

[15] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose, "Understanding error propagation in GPGPU applications," in *Proc. SC*, Salt Lake City, UT, USA, Nov. 2016, pp. 240–251.

[16] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *Proc. ISPASS*, Santa Rosa, CA, USA, Apr. 2017, pp. 249–258.

[17] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda, "Statistical fault injection," in *Proc. DSN*, Anchorage, AK, USA, Jun. 2008, pp. 122–127.

[18] S. E. Michalak *et al.*, "Assessment of the impact of cosmic-ray-induced neutrons on hardware in the roadrunner supercomputer," *IEEE Trans. Device Mater. Rel.*, vol. 12, no. 2, pp. 445–454, Jun. 2012.

[19] C.-Y. Cher, M. S. Gupta, P. Bose, and K. P. Muller, "Understanding soft error resiliency of blue gene/Q compute chip through hardware proton irradiation and software fault injection," in *Proc. SC*, New Orleans, LA, USA, Nov. 2014, pp. 587–596.

[20] C. Bottoni *et al.*, "Heavy ions test result on a 65nm SPARC-V8 radiation-hard microprocessor," in *Proc. IEEE IRPS*, Waikoloa, HI, USA, Jun. 2014, pp. 5F.5.1–5F.5.6.

[21] P. Greenhalgh, "Big.LITTLE processing with ARM cortex-A15 & cortex-A7," ARM, White Paper, 2011.

[22] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proc. HPCA*, Raleigh, NC, USA, Feb. 2009, pp. 117–128.

[23] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual: User-level ISA, version 2.1," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-118, 2016, vol. 1.

[24] *RISC V cores and SoCs*. Accessed: Jul. 22, 2018. [Online]. Available: https://riscv.org/risc-v-cores/

[25] Y. Lee *et al.*, "An agile approach to building RISC-V microprocessors," *IEEE Micro*, vol. 36, no. 2, pp. 8–20, Mar. 2016.

[26] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley out-of-order machine (BOOM): An industrycompetitive, synthesizable, parameterized RISC-V processor," Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-167, 2015.

[27] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *Proc. DSN*, Florence, Italy, Jun./Jul. 2004, pp. 61–70.

[28] Q. Lu, G. Li, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, "Configurable detection of SDC-causing errors in programs," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 3, pp. 88:1–88:25, Mar. 2017, doi: 10.1145/3014586.

[29] Y. Zhang, J. W. Lee, N. P. Johnson, and D. I. August, "DAFT: Decoupled acyclic fault tolerance," in *Proc. PACT*, Vienna, Austria, 2010, pp. 87–98.

[30] K. Pattabiraman, G. P. Saggese, D. Chen, Z. Kalbarczyk, and R. K. Iyer, "Automated derivation of application-specific error detectors using dynamic analysis," *IEEE Trans. Depend. Sec. Comput.*, vol. 8, no. 5, pp. 640–655, Sep./Oct. 2011.

[31] J. Bachrach *et al.*, "Chisel: Constructing hardware in a Scala embedded language," in *Proc. DAC*, San Francisco, CA, USA, Jun. 2012, pp. 1212–1221.

[32] A. KleinOsowski and D. J. Lilja, "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research," *IEEE Comput. Archit. Lett.*, vol. 1, no. 1, p. 7, Jan./Dec. 2002.

[33] C. Wang, H.-S. Kim, Y. Wu, and V. Ying, "Compiler-managed software-based redundant multi-threading for transient fault detection," in *Proc. CGO*, San Jose, CA, USA, Mar. 2007, pp. 244–258.

[34] J. Suh, M. Annavaram, and M. Dubois, "PHYS: Profiled-hybrid sampling for soft error reliability benchmarking," in *Proc. DSN*, Budapest, Hungary, Jun. 2013, pp. 1–12.

[35] X. Fu, J. Poe, T. Li, and J. A. B. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," in *Proc. MASCOTS*, Monterey, CA, USA, Sep. 2006, pp. 147–155.

[36] *Cobham Gaisler Leon3 Processor*. Accessed: Jul. 22, 2018. [Online]. Available: http://gaisler.com/leon3

[37] G. Bronevetsky, B. de Supinski, and M. Schulz, "A foundation for the accurate prediction of the soft error vulnerability of scientific applications," Lawrence Livermore Nat. Lab., Livermore, CA, USA, Tech. Rep. LLNL-CONF-410635, 2011.

[38] 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 *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Sep. 2003, pp. 29–40.

[39] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *Proc. ISCA*, San Diego, CA, USA, 2007, pp. 460–469.

[40] B. Wibowo, A. Agrawal, and J. Tuck, "Characterizing the impact of soft errors across microarchitectural structures and implications for predictability," in *Proc. IISWC*, Seattle, WA, USA, Oct. 2017, pp. 250–260.

[41] S. Mirkhani, B. Samynathan, and J. A. Abraham, "In-depth soft error vulnerability analysis using synthetic benchmarks," in *Proc. VTS*, Napa, CA, USA, Apr. 2015, pp. 1–6.

[42] S. K. Sastry Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Application resiliency analyzer for transient faults," *IEEE Micro*, vol. 33, no. 3, pp. 58–66, May 2013.

[43] S. K. S. Hari, R. Venkatagiri, S. V. Adve, and H. Naeimi, "GangES: Gang error simulation for hardware resiliency evaluation," in *Proc. ISCA*, Minneapolis, MN, USA, Jun. 2014, pp. 61–72.

[44] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: Probabilistic soft error reliability on the cheap," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Pittsburgh, PA, USA, Mar. 2010, pp. 385–396.

[45] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *Proc. DSN*, Jun. 2014, pp. 375–382.

[46] H. Cho, C.-Y. Cher, T. Shepherd, and S. Mitra, "Understanding soft errors in uncore components," in *Proc. DAC*, San Francisco, CA, USA, 2015, Art. no. 89.

HYUNGMIN CHO received the B.S. degree in computer science engineering from Seoul National University, South Korea, in 2005, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, in 2010 and 2015, respectively. He was a Research Scientist with Intel Labs, Santa Clara, CA, USA. He is currently an Assistant Professor with the Department of Computer Engineering, Hongik University, South Korea. His research interests include reliable computer systems and computing model for robust systems.