Characterizing Soft Error Vulnerability of CPUs Across Compiler Optimizations and Microarchitectures

George Papadimitriou Dimitris Gizopoulos
Department of Informatics and Telecommunications
University of Athens, Greece
{georgepap | dgizop}@di.uoa.gr

***Abstract*—In this paper, we present a fine-grained characterization of the impact of transient faults (soft errors) on program execution for different compiler optimization levels and two out-of-order microarchitectures through extensive microarchitecture-level fault injection experiments. We evaluate how the different levels of compiler optimization impact the failure probability of the most important hardware structures in two different out-of-order Arm microarchitectures (Cortex-A15 and Cortex-A72). We analyze 32 different executables: sources come from eight different benchmarks with large datasets, each one compiled with three different levels of compiler optimization (O1, O2, O3) and the baseline unoptimized code level (O0); execution times of the 32 binaries range from 72M cycles to 1.4B cycles. We show how the different compiler optimization levels affect the vulnerability of eight important hardware structures. We perform extensive soft error fault injection campaigns to measure with high statistical significance the Architectural Vulnerability Factor (AVF) of all hardware structures at each optimization level, and identify the structures whose vulnerability is more sensitive to compiler optimizations. Finally, we aggregate the vulnerabilities of the hardware structures into the overall failure rates of the microprocessor and complement with a performance-aware comparison of all optimization levels. The performance-aware vulnerability analysis shows that higher optimization levels counterbalance their increased vulnerability with the speedup the deliver. From the failure rates sole point of view, an unprotected design has variable behavior, however, when typical ECC protection is employed the O2 optimization level is consistently the most robust one, while for more recent microarchitectures, O1 can be equally robust to O2 which is not the case in older microarchitectures.**

***Index Terms*—reliability; microprocessors; transient faults; microarchitecture; compilers; architectural vulnerability factor; microarchitecture-level fault injection**

I. Introduction

The extreme scaling of technology has a negative impact on the reliable operation of the microprocessors, making them more vulnerable to cosmic radiation, manufacturing defects, and low-voltage operation [1]-[5]. As a result, transient faults (soft errors) tend to appear more frequently than in previous manufacturing generations, which, in turn, makes them a major threat to reliable computing system operation. Several metrics have been proposed to express the system vulnerability against transient faults. The Architectural Vulnerability Factor (AVF) [6] was proposed by Mukherjee, *et al*. to quantify the microprocessor's vulnerability to transient faults. The AVF of a microprocessor hardware structure is the probability that a fault in it will affect the correct execution of a program.

The resilience of a microprocessor and program combination to soft errors can be affected by many factors, including the programming language, the algorithm of the workload and its inputs, the environmental conditions (e.g., altitude, temperature), the voltage, and the aging of the microprocessor hardware [7] [8]. Among these factors, optimizations introduced by compilers modify the way that a program utilizes the hardware structures of the microprocessor, even if the source algorithm of the program and the inputs remain the same. Recent studies have considered the impact of compiler optimizations on microprocessor's vulnerability [9]-[15]. These studies, however, have either kept the analysis at the software layer without any consideration of the underlying microarchitecture or have only investigated very few hardware structures (such as the register file only) and not the entire CPU. A fine-grained analysis for all hardware structures when the full system stack is employed is very important and such studies are crucial not only for early reliability assessment by hardware designers, but also for software resiliency, since compiler optimizations are one of the most successful ways to improve performance with relatively low effort by software engineers.

It has been shown that the resilience to transient faults varies dramatically across different applications, and across different phases of the same application [16] [17]. Compiler optimizations, also affect the vulnerability of the application [18] [19] [20] [21]. For example, code optimization can increase the hardware structures utilization and the instruction throughput of out-of-order processors, which generally improve performance. On the other hand, since transient faults occur randomly in the microprocessor structures, higher utilization implies higher sensitivity to these faults, as there are more microprocessor's components that operate on application's instructions and data in parallel. Additionally, increasing data locality may reduce the number of cache misses, and thus, faults occurring in a cache memory may have lower probability to propagate themselves to lower memory hierarchy levels, but may increase the time a certain value resides in a register before being changed. Thus, a fault in the microprocessor register may have higher probability of propagating into the application's data structures and generate data corruptions or crashes.

In this paper, we investigate the impact of compiler optimization levels and the microarchitecture on microprocessor vulnerability to soft errors. We study the three different compiler optimization levels of GCC compiler (O1, O2, O3) and compare them to the unoptimized code (O0), to thoroughly identify their vulnerability differences for all important structures of an out-of-order microarchitecture. To do so, we employ fine-grained, statistically significant microarchitecture-level fault injection to measure the reliability of each optimization level across several different workloads and two different Arm

microarchitectures (i.e., a Cortex-A15 and a Cortex-A72). Statistical Fault Injection (SFI) [22] [23] [24] is often the most time-consuming reliability evaluation method, as it requires multiple simulations in order to gather a statistically significant sample. However, it is the most accurate method and provides insights on the fault effects, as it experimentally determines the vulnerability of a system and the fine-grain effect of every single injected fault [25]. To the best of our knowledge, this is the first work that provides a fine-grained analysis using SFI at the detail of the microarchitecture level and individually analyzes the impact of the compiler optimization levels on the vulnerability of each individual hardware structure of two different microarchitectures and the microprocessor as a whole. Specifically, the contributions of this paper are:

1. We measure the AVF of the 8 most important microprocessor hardware structures including all their subfields/arrays (L1 data and instruction caches (tags and data fields), L2 cache (tags and data fields), the physical register file, the load and store queues, the issue queue (Source and Destination fields), and the reorder buffer (all its four fields) for the three compiler optimization levels (O0, O1, O2, O3) of 8 different benchmarks. The evaluation is performed for two different Arm microarchitectures (an older 32-bit Cortex-A15 and a newer 64-bit Cortex-A72). In total, we evaluate 960 combinations of programs + hardware structures, through 1,920,000 end-to-end fault injection runs. We employ the *large* input datasets in all benchmarks with end-to-end execution, so that all hardware structures are sufficiently utilized; this decision significantly increases the total simulation time but is important for analyzing the executions of realistic programs.
2. We thoroughly analyze the impact of different compiler optimizations and microarchitectures on the vulnerability of each individual microprocessor structure. We show how each optimization level *significantly* affects the vulnerability of each microarchitectural structure and provide essential observations about the magnitude of different fault effect classes, and how the levels of these classes can be altered on each hardware structure when different optimization levels are applied to the source code of a program.
3. We present the overall impact of compiler optimization levels on each individual microprocessor structure of each microarchitecture. Since each combination of a hardware component and a benchmark has a different AVF, we employ an aggregating weighted AVF metric, which takes into consideration the AVF differences among different workloads and provides an overall AVF for each structure that takes into consideration the different execution times.
4. We finally present the Failures in Time (FIT) rate of the entire out-of-order microprocessors (failures per 10^9 hours of operation). By using this rate, we can further extend our observations to analyze the impact of the different optimization levels for each microarchitecture on the total vulnerability of the microprocessor (not only individual hardware structures). In addition, we introduce a performance-aware metric (failures per execution) which compares all optimization levels, to identify the best tradeoff between the performance improvement that optimizations deliver and the impact they have on the entire microprocessor reliability.

II. Background and Related Work

A. Compiler Optimization Flags

Modern compilers, like GNU Compiler Collection (GCC), are the key tool to unlock the performance of applications running on modern microprocessors. To generate an effective machine code from the high-level source code, the compiler usually requires many iterations (passes) to rearrange the source code into a basic instruction sequence and determines the instruction sequence which can provide improved performance or reduced code size. Since the large range of compiler optimizations will greatly affect the performance of applications, there should also be a range of features in each application which can be affected by the compilation process. These include loops, nested loops, different arithmetic types, calls to procedures, bitwise operations, vector/array accesses, etc.

GCC includes a large variety of optimizations, which may be individually or in groups turned on. In order to achieve better performance, GCC can enable plenty of optimizations, which are called compiler optimization levels. The most common and essential optimization levels are denoted through the O0, O1, O2, and O3 compilation flags. Level O0 indicates that no optimization will be enabled during the compilation of the source code. Higher optimization levels add extra optimizations in the source code compared to the lower ones and perform more global transformations on the program by applying more sophisticated analysis algorithms to generate a faster and more compact code [26] [27]. The final effects of the compilation time and the resulting improvement in execution time depend on the particular application and the underlying hardware design. From all available optimizations, some of them reduce the size of the resulting machine code, while others try to create a faster code, potentially increasing its size (e.g., loop unrolling increases the machine code size, but typically reduces the execution time). Furthermore, each optimization level affects in different ways the instruction execution flow and the resulting behavior of each individual microprocessor component.

The effects of each optimization level explored in this work are briefly described below:

O0: the compiler does not perform any optimization and each command in the source code is converted directly to the corresponding machine code.

O1: the compiler aims to reduce the code size and execution time. It enables instruction reordering, loop optimization, and other common optimizations.

O2: this level further enhances O1 optimization, including instruction scheduling, recursive call optimization, and cross jumps.

O3: this level turns on more sophisticated optimizations, including procedure inlining, moving loop invariant conditions out of the loop, and other optimizations that further enhance the performance of the executed code but the generated code is usually larger than the other optimization levels.

B. Related Work

Previous studies focused on investigating the impact of compiler optimizations on microprocessor's vulnerability by primarily using software-based tools for reliability evaluation, i.e., assuming that a software visible location (an architectural register) is the starting point of the analysis. Although these studies, which use software-based techniques to investigate the impact of compiler optimizations, allow reliability assessment on long and realistic workloads, they do not capture the real hardware vulnerability, since the starting point of the experiment is a corrupted instruction (not a hardware structure that is hit by a particle and suffers a soft error) [25]. Moreover, to the best of our knowledge, there are no previous studies that comprehensively investigate the impact of different compiler optimization levels on the major microprocessor components of different microarchitec-

tures, e.g., all cache memory levels, and the fundamental pipeline structures, such as Reorder Buffers, Physical Register file, the Load/Store and Issue Queues – our analysis covers all these structures. In particular, Links *et al*. in [9] investigate the effects of compiler optimizations on an embedded Arm Cortex-A9 microprocessor for the Register File only by using interrupt-based fault injection and heavy ion experiments. Demertzi, *et al*. in [10] provide coarse-grain estimations (not through fault injection) about the AVF of the Instruction Store Queue, the Load Store Queue, and the Reorder Buffer, by using some estimation equations. Based on an estimated AVF for these three structures, the authors explain how the compiler optimization levels affect their vulnerability. In this study, we go far beyond this and investigate not only the impact of optimizations on 8 major microprocessor components (for all their fields), but also provide a fine-grained analysis of the vulnerability for each individual component across different benchmarks and for the microprocessor as a whole and for different microarchitectures. This way we present a thorough evaluation of the impact of compiler optimizations on the vulnerability of different microarchitectures.

Jones, *et al*. in [11] evaluate the effects of compiler optimizations on the reliability of the microprocessor, by using ACE analysis, as initially described in [6]. The impact of the compiler optimizations in the AVF is evaluated by trying to reduce the AVF-delay-square-product (ADS). The authors primarily focus on the tradeoffs between vulnerability and performance for each optimization level, without analyzing their behavior on each microprocessor component. Although ACE analysis aims to profile the data lifetime inside the hardware structures and quantify their exposure to estimate the vulnerability, it does not provide fine-grained insights on the fault effects and is known to be pessimistic (i.e., over-estimates the vulnerability of a microprocessor structure) [23]. Cook, *et al*. in [28] use an instruction-level injection technique to demonstrate how an incorrect architectural state can result in a correct program behavior. Narayanamurthy, *et al*. in [12] use LLVM-based fault injection to examine the effect of compiler optimizations on the error resilience of the software. Ashraf, *et al*. in [19] analyze how the most common compiler optimization levels impact the vulnerability of several applications, the trade-offs between performance and vulnerability, and the relations between compiler optimization and application vulnerability. Similar to the previous works, they use a software-level metric to show that highly optimized code is generally more vulnerable than unoptimized code. Sangchoolie, *et al*. in [13] examine the effect of compiler optimization levels on the SDC rates of different workloads for architectural registers and main memory (again software visible locations). They use instruction-based fault injection and find that the optimization levels reduce the resilience of the application. Bergaoui and Leveugle in [14] analyzed the impact of compiler optimizations on data reliability in terms of variable liveness. Variable liveness is the time period between the variable that is written and it is last read before a new write operation. The authors state that the liveness is not related only to compiler optimizations, but also on the application itself. Ferreira, *et al*. in [15] focused on compiler optimizations of embedded software and reported that random selection of optimizations, can decrease software reliability, when the applications were developed to detect and correct radiation-induced control-flow errors.

Although PVF, LLVM-based and instruction-level fault injection techniques quantify the architecture-level fault masking inherent in a program, they do not consider the vulnerability of the actual underlying hardware, and thus, can lead to misleading findings and observations [25]. In contrast to these studies, in this work, we are based on *detailed* microarchitecture-level reliability evaluation and provide extensive fault injection campaigns to reach statistically safe results and accurately support our observations.

III. Experimental Methodology

In this work, we use an extensive microarchitecture-level fault injection simulation-based study and collect a set of important observations for vulnerability analysis of a microprocessor in different compiler optimization levels. For this study, we select 8 representative benchmarks from the MiBench benchmarks suite [29], with the largest possible input data sets, and have simulated them for 8 major hardware components for two different microarchitectures. This corresponds to 1,920,000 fault injections for all components, benchmarks, and microarchitectures used in this study. The simulated microprocessors are (i) an Arm Cortex-A15 out-of-order model (Armv7 ISA) and (ii) an Arm Cortex-A72 out-of-order model (Armv8 ISA).

A. Fault injection Platform

Among the available abstraction models, microarchitecture-level fault injection is the only one that comes with sufficient hardware detail and can run full-system simulation including the operating system [25]. Further, microarchitecture-level fault injection offers high observability, allowing fine-grained selection of where exactly the faults 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 the effect of the fault on the entire system stack as we describe in the following subsection. Our microarchitectural modelling is based on gem5 simulator, the state-of-the-art, flexible full-system cycle-accurate simulator [30]. Table I presents the configuration of gem5 for the two microprocessor models.

For the reliability assessment we use GeFIN [31]: a microarchitecture-level fault injection framework that was built on top of the gem5 simulator. The faults are injected in the 8 most important components (including all their subfields) that per our estimations correspond to 90% – 95% of the memory cells of the microprocessor: L1 data and instruction caches (tags and data fields), L2 cache (tags and data fields), the physical register file, the load and store queues, the issue queue (Source and Destination fields), and the Reorder Buffer (all its four fields). For each of the 8 components and each of their fields (sub-components), 2,000 single-bit faults were randomly injected (following the uniform distribution as defined in [22]), resulting in 1,920,000 fault injections for the two microarchitectures, all 8 benchmarks and the 4 different optimization levels. We follow the widely adopted formulation of [22] for the statistical fault sampling

Table I. Microprocessors Configuration.

Parameter	Cortex-A15	Cortex-A72
ISA	Armv7	Armv8
Pipeline	Out-of-Order	Out-of-Order
L1 Data Cache	32 KB (2-way), PIPT*	32 KB (2-way), PIPT
L1 Instruction Cache	32 KB (2-way), PIPT	48 KB (3-way), PIPT
L2 Cache	1 MB (8-way), PIPT	2 MB (16-way), PIPT
Physical Register File	128 registers	192 registers
Issue Queue	32 entries x 32 bit	64 entries x 64 bit
Load / Store Queue	16 entries x 32 bit	16 entries x 64 bit
Reorder Buffer	40 entries	128 entries
Fetch width	3	3
Execute Width	6	6
Writeback Width	8	8

** PIPT = Physically-Indexed Physically-Tagged*

calculations; 2,000 fault injections per component correspond to a very small 2.88% error margin with very high 99% confidence level.

B. Benchmarks Selection and Characteristics

For our experiments we used 8 benchmarks from the MiBench benchmarks suite [29]. The suite is commonly used in many reliability studies [2] [3] [11] [31]-[37]. The suite includes realistic benchmarks from diverse application domains that share data and control flow characteristics with SPEC benchmark suite [38]. To maintain a large breadth in the analysis and avoid the bias of results on specific applications, we have chosen the *largest* available input datasets for all benchmarks. The benchmarks were also chosen to cover a wide variety of different computational characteristics that may affect the program execution metrics, such as vulnerability, execution time, and interaction-sensitivity to compiler optimizations. The main characteristics that have been taken into consideration for the selection of benchmarks were (1) the integer operation intensity, (2) the frequency of branches, (3) the memory-hierarchy accesses frequency, and (4) the magnitude of performance improvement when applying any optimization level to the source code of the benchmark. Benchmarks with these characteristics can provide essential diversity to the executed machine code and will lead to comprehensive observations.

C. Fault Effects Classification

The GeFIN classifies the outcomes of each fault injection based on the impact of the fault on the simulated system, as follows:

Silent Data Corruption (SDC): The simulation finished normally, but the program output was different compared to the fault-free simulation, without any observable indications of this effect.

Timeout: The simulation did not finish within a certain amount of time, equal to two times the fault-free execution time. These simulations are externally stopped to resolve potential deadlock or livelock situations.

Crash: A simulation that did not reach the end of the execution, as it was disturbed by a catastrophic event. As a result, no program output was produced. A crash may refer to a process crash (killed process) or a system crash (kernel panic).

Assert: The simulation was unexpectedly terminated due to a high-level condition that the simulator was unable to handle. If, for instance, a physical address that is outside the system map is requested, the simulator cannot tell how a real system would behave and raises an assertion.

When the simulation was executed with no deviations from the fault-free simulation, the injected fault is categorized as a **Masked** fault. The result of a masked simulation is identical to the golden (fault-free) simulation, and thus, we do not present it in the diagrams.

IV. Compiler Optimization Effects on AVF

In this section, we summarize the performance improvement that each compiler optimization level delivers to our benchmarks and describe the compiler optimization and microarchitecture effects on AVF. Note that the scope of this work is not to analyze the reasons why each different application code provides different vulnerability results (this would suggest a detailed dynamic analysis of each application), but the effects of different optimization levels and microarchitectures on the microprocessors vulnerability. To this end, to comprehensively summarize the detailed data and results for each hardware component and show the aggregated AVF between Cortex-A15 and Cortex-A72, we average the AVF of each component across the 8 different benchmarks. To account for the different execution times of the benchmarks, instead of calculating the straightforward arithmetic mean of the AVFs of the component for the different benchmarks, we *weight* the AVFs with the execution time of the benchmarks, similarly to [25] and [39]. Thus, very short benchmarks will have a smaller impact on the component's aggregate AVF compared to longer ones. The weighted AVF is calculated by summing the AVF of all benchmarks, each multiplied by the execution time of the corresponding benchmark and dividing them by the sum of the execution times of all the benchmarks, as shown in equation (1):

$$\text{wAVF(c)} = \sum_{k=1}^{N}(AVF_k(c) \times t_k) \;/\; \sum_{k=1}^{N} t_k \qquad (1)$$

where, *wAVF(c)* is the weighted AVF of a component *c*, $AVF_k(c)$ is the AVF of component *c* for benchmark *k*, t_k is the execution time of each benchmark *k*, and *N* is the total number of benchmarks. The weighted AVF (wAVF) is shown for each optimization level at the rightmost bars of each figure.

A. Performance Analysis

The main target of compilers is either to increase the performance of an executed code or to reduce the generated code size. However, the magnitude of performance improvement of each optimization level varies significantly across different benchmarks, as shown in Fig. 1 for the benchmarks of our work. Optimizations on some benchmarks, such as *qsort* and *patricia* provide only slight performance improvement, while others, such as *fft* provide nearly zero improvements. The inability to provide effective optimization in these benchmarks, is due to the structure of the code. On the other hand, *dijkstra* and *gsm* can be heavily optimized by all optimization levels. There are few library-calls in these benchmarks, and many structures in the code (e.g., recursive procedures, loops, etc.) that the compiler can effectively manage to exploit in order to provide essential optimizations in the generated code. Overall, as Fig. 1 shows, all benchmarks have different response to each different optimization level. Optimization level O1 provides the largest improvements in the most cases, while higher optimization levels sometimes increase but they may also decrease the performance compared to O1 (see *gsm*). Moreover, in all cases (except *sha*) the optimization level O3 shows marginally worse performance compared to O1 and O2, of 3.5% at most. This observation confirms the fact that optimization efficiency depends on source code structure. Note that, while in Cortex-A72 the absolute performance is higher than in Cortex-A15 for all benchmarks, the relative performance among different optimization levels shown in Fig. 1 is the same between the two microarchitectures.

B. L1 Instruction Cache (L1I)

Fig. 2 presents the AVF results of L1I cache for the Data and Tag

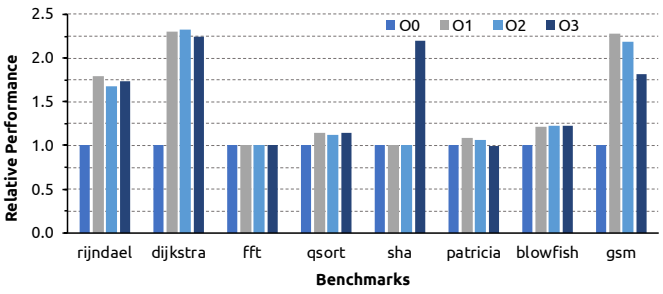

Fig. 1. Relative performance among all optimization levels.

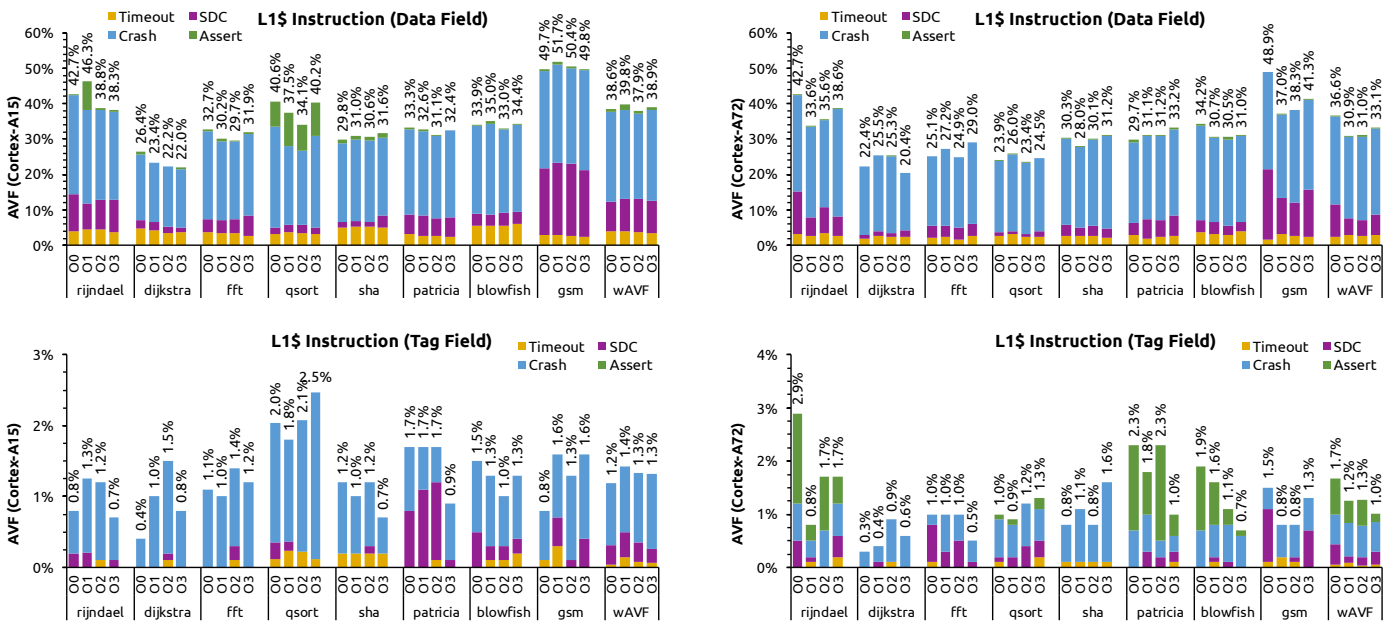

Fig. 2. AVF for L1 Instruction Cache (Data field at the top and Tag field at the bottom) for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

fields, for all benchmarks and all optimization levels for both microarchitectures used in this study. The foremost observation is that the most prominent faulty behavior among the vulnerable cases for any benchmark and any optimization level (including O0) for both microarchitectures is the Crash fault class. The reason is that faults in the L1I cache affect the instructions and immediate values and very likely lead to a crash. Compared to Crashes, the SDC fault behavior is extremely low for most of the benchmarks, especially for optimization levels O1, O2, and O3. Comparing the two microarchitectures, we can see in the weighted AVF graphs (wAVF – the rightmost bars for each diagram) that for both Data and Tag fields of the L1I cache, the vulnerability of any optimization level is significantly reduced in Cortex-A72, while for Cortex-A15 the vulnerability of any optimization level is virtually the same (for Data field) and is slightly increased for the Tag field. Specifically, for Cortex-A72 the vulnerability of optimization levels for Data field is up to 18.4% less than the unoptimized code, while for Tag field is up to 70% less vulnerable than the unoptimized code (O0). As we can also see in Fig. 2 (top-right and bottom-right diagrams), although different optimization levels may have different behavior, it is clearly shown that the main reason for the vulnerability reduction of higher optimization levels in L1 Instruction cache of the Cortex-A72 is the reduction of Crash and SDC fault classes.

Another observation is that the vulnerability for each optimization level between the two microarchitectures (for both the Data and the Tag fields) is virtually at the same levels. For example, if we see the wAVF for each graph, we observe that the vulnerability percentage between the two microarchitectures is very close, although as we discussed earlier, the compiler optimizations provide significantly less vulnerability in Cortex-A72 compared to O0.

C. L1 Data Cache (L1D)

Fig. 3 presents the AVF results of L1 Data Cache for the Data and Tag fields for all benchmarks and all optimization levels for both microarchitectures used in this study. The foremost observation in case of L1D cache for both microarchitectures, is that the most prominent faulty behavior among the vulnerable cases for the most of benchmarks and optimization levels (including O0) is the SDC fault class. The reason is that faults in L1 data cache affect the actual data words of an application, so there is high probability for a fault to corrupt the calculations, and thus, the final results of an application.

Unlike the L1 instruction cache (in which both microarchitectures show very close vulnerabilities for Data and Tag Fields), L1 data cache shows that Cortex-A15 has significantly less vulnerability than Cortex-A72 for both Data and Tag Fields. Specifically, the wAVF bars for O0 show that Cortex-A72 is 32.3% more vulnerable than Cortex-A15 for the Data field, while for the Tag field is 45.2% more vulnerable. Another important observation, is that the vulnerability of different compiler optimization levels is very close to the unoptimized code for both microarchitectures for Data field. However, Cortex-A72 shows slightly reduced vulnerability in all optimization levels for the Tag field, which is up to 8.6% less than the O0.

D. L2 Cache (L2)

Fig. 4 presents the AVF results of L2 cache for the Data and Tag fields for all benchmarks and all optimization levels for both microarchitectures used in this study. The foremost observation in case of L2 cache, is that the most prominent faulty behavior among the vulnerable cases for all benchmarks and all optimization levels (including O0) across both microarchitectures is the SDC fault class, while only for *dijkstra* the most prominent faulty behavior is the Crash fault class for all optimization levels for the Data field. Similar to L1 data cache, and unlike the L1 instruction cache, Cortex-A72 is significantly more vulnerable in L2 cache compared to Cortex-A15 for any optimization level. Specifically, the vulnerability of Cortex-A72 is up to 32.4% more than the vulnerability of Cortex-A15 for the Data field, and up to 45.2% more for the Tag field. Although the absolute vulnerability is higher in Cortex-A72 than in Cortex-A15, the vulnerability of different optimization levels for any microarchitecture of both Data and Tag fields is virtually the same. However, for the Tag

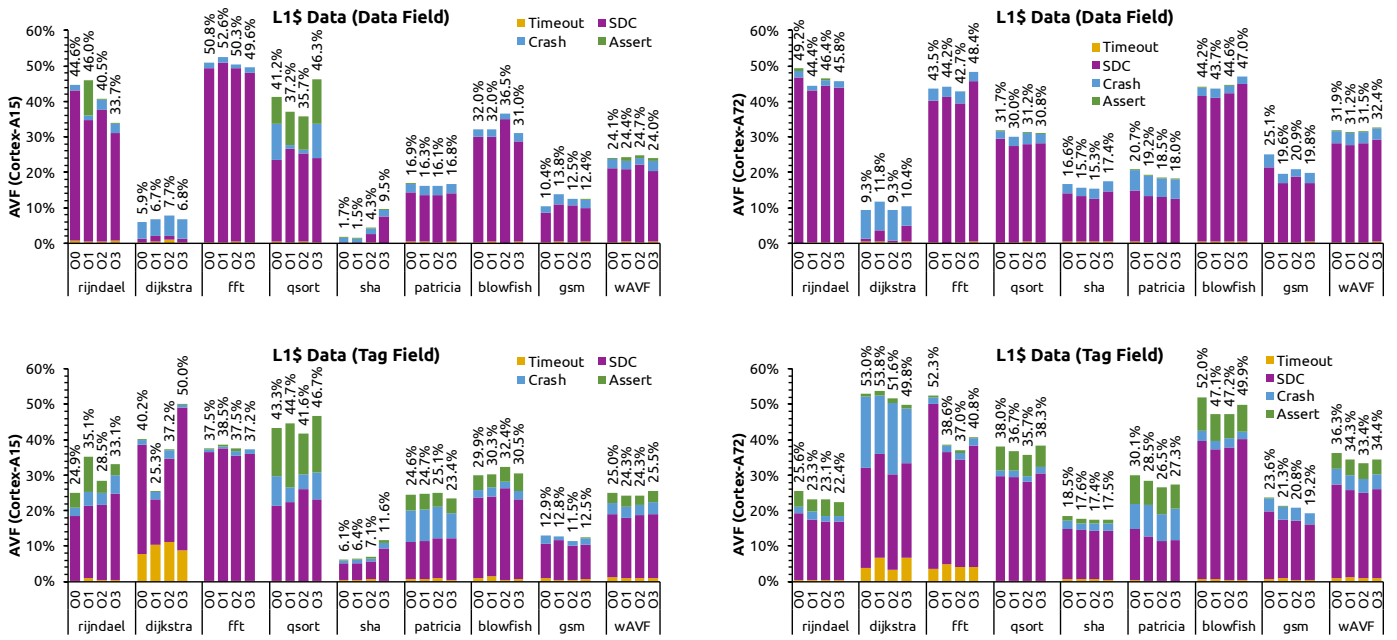

Fig. 3. AVF for L1 Data Cache (Data field at the top and Tag field at the bottom) for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

field of Cortex-A72, the vulnerability of compiler optimization levels is 8.6% less than the unprotected core (O0). This is in line with the L1 data cache we discussed earlier.

E. Physical Register File (RF)

Fig. 5 presents the AVF results of Physical Register File (RF) for all benchmarks and all optimization levels for both microarchitectures used in this study. The foremost observation in case of RF, is that in most cases the optimized code is more vulnerable compared to the unoptimized one (O0). Compared to the caches' vulnerability analysis of the previous subsections, this means that the fundamental modifications induced by compiler in all optimization levels have a major impact on the registers and not on the caches' levels. The AVF results presented in Fig. 5, show that compiler optimizations in the source code of an application can lead to higher vulnerability in the physical register file than in a cache memory array. This can be attributed to the fact that compiler optimization methods try to increase the register file utilization (i.e., higher read and write operations), as much as possible, to minimize the memory accesses. However, as it is shown, the optimizations induced by the compiler, despite the improved performance, they increase the vulnerability of the physical register file. For example, *blowfish* is 2.25× more vulnerable at O1

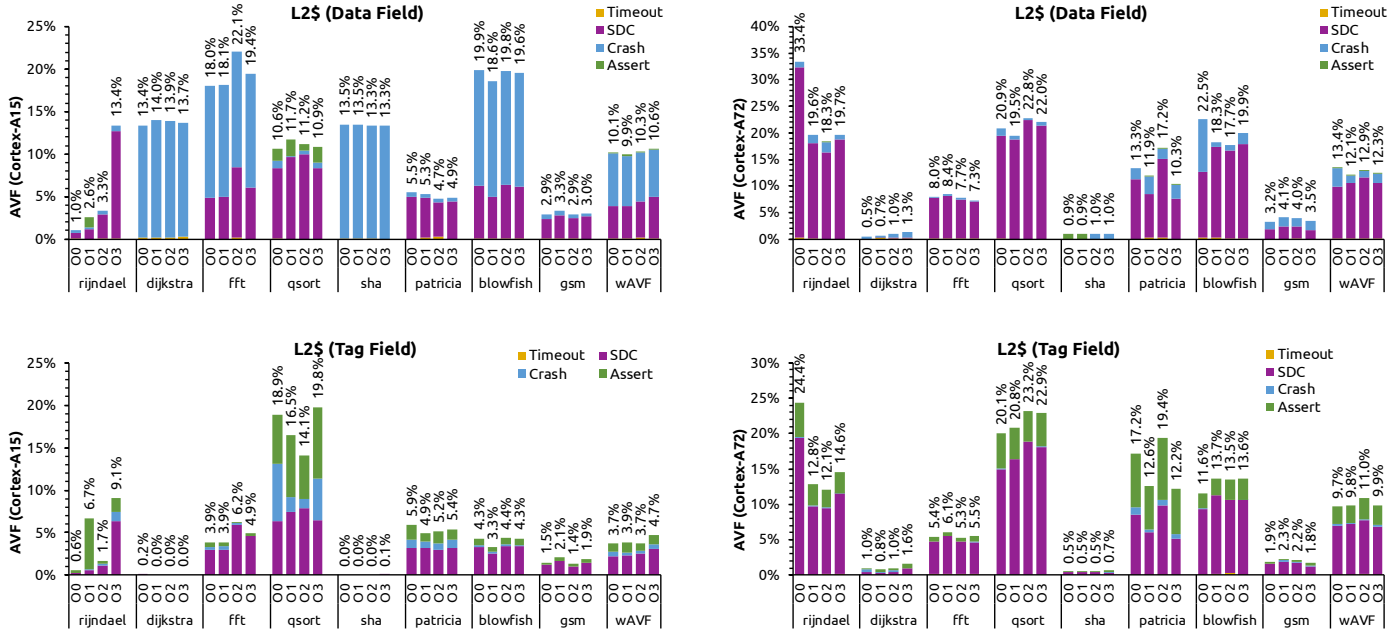

Fig. 4. AVF for L2 Cache (Data field at the top and Tag field at the bottom) for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

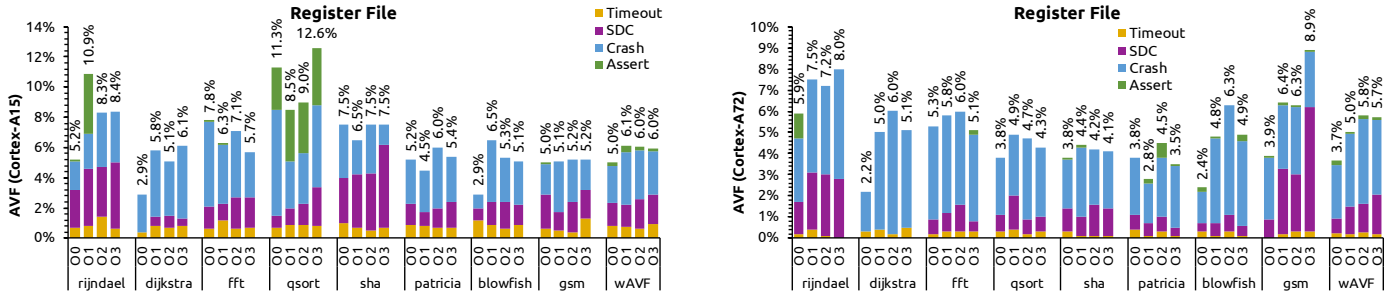

Fig. 5. AVF for Physical Register File for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

level, compared to O0 for the Cortex-A15, and 2× more vulnerable for Cortex-A72. To further support this statement, we observed that there is a significant increase of registers utilization during the execution of optimized code. For example, *dijkstra*'s register utilization is increased 4× in O1 compared to the unoptimized code. Other benchmarks, such as *sha* and *patricia* have nearly zero difference in register file utilization among different optimization levels. Another interesting observation is that in all benchmarks, Crash and SDC fault classes are balanced for each optimization level (including O0), although in most cases both the Crash and SDC classes are slightly increased in O1, O2, and O3 compared to O0. Interestingly, compiler optimization levels show higher impact on the vulnerability of Cortex-A72 compared to Cortex-A15. Specifically, the vulnerability difference for Cortex-A72 is up to 56.7% higher for the optimized codes, while for Cortex-A15 is up to 22% higher compared to the unoptimized code (O0).

F. Load and Store Queues (LQ and SQ)

We present the behavior of the Load and the Store Queues together, as they have very similar behavior regarding the impact of each optimization level on AVF. Fig. 6 presents the AVF results of Load Queue (we omit the graph for SQ due to space limitations, but we use the data in the calculations for the overall microprocessor) for all benchmarks and all optimization levels for both microarchitectures used in this study. The foremost observation in case of the LQ/SQ, is that the most prominent faulty behavior among the vulnerable cases is the Assert fault class for all optimization levels. The reason is that LQ and SQ contain register operands, meaning that any corruption in the register operands will certainly lead to an unexpected (unhandled) microprocessor operation, since it is highly likely for an unused or invalid register to appear due to the corruption. However, in the case of the LQ/SQ we can observe in Fig. 6 that the Assert fault class is the only class that affects these components. As we can see in Fig. 6, while the absolute vulnerabilities (wAVF) of the Cortex-A72 are lower than the Cortex-A15, for both LQ and SQ, the compiler optimization levels of Cortex-A72 show significantly higher vulnerability than the unoptimized code, which is not the case for Cortex-A15 where all optimization levels provide virtually the same vulnerability results. Specifically, in Cortex-A72 the vulnerability of optimized codes is up to 19.4% higher than the unoptimized code (O0). Similar to the Register File, in Cortex-A72 we observe significantly higher vulnerability of all optimization levels than in Cortex-A15.

G. Issue Queue (IQ)

Fig. 7 presents the AVF results of Issue Queue for the Source field for all benchmarks and all optimization levels for both microarchitectures used in this study (we omit the graph for Destination field due to space limitations, but we use the data in the calculations for the overall microprocessor). Unlike all other microprocessor structures, the Issue Queue is the only one that provides increased levels of Timeout and Assert fault classes. More specifically, as we can see in Fig. 7, the vulnerability of the Assert and Timeout fault classes are virtually balanced for any benchmark and microarchitecture, while all other fault classes provide nearly zero vulnerability. Another interesting observation is that in Issue Queue the vast majority of benchmarks in Cortex-A15 show higher AVF at O0 compared to other optimization levels, while in most of the cases the optimization levels show lower vulnerability compared to the O0. On the contrary, in Cortex-A72 we can see that while the absolute vulnerability is lower than the Cortex-A15 (by up to 70% lower for the O0), all optimization levels provide higher vulnerability compared to the unoptimized code (O0), which is up to 20% higher. This observation is in line with the Register File and Load and Store queues, as we discussed.

H. Reorder Buffer (ROB)

Fig. 8 presents the AVF results of Reorder Buffer for the PC field for all benchmarks and all optimization levels for both microarchitec-

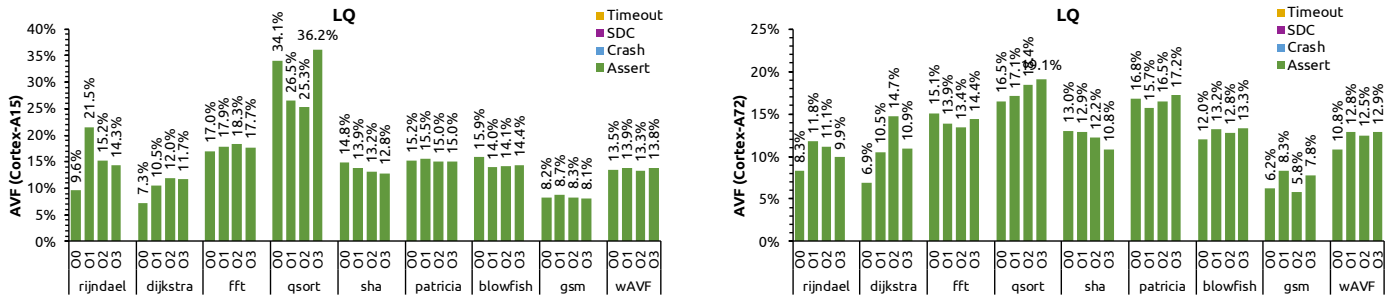

Fig. 6. AVF for Load Queue (top diagrams) and Store Queue (bottom diagrams) for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

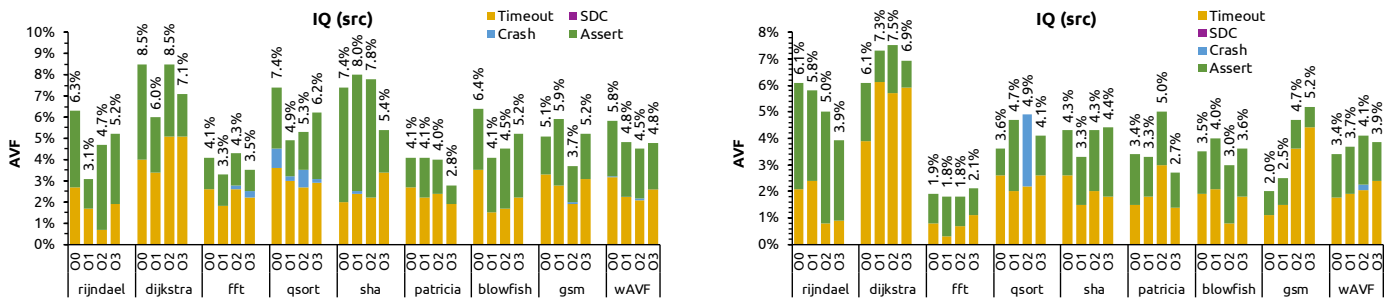

Fig. 7. AVF for Issue Queue for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

tures used in this study (we omit the graph for the other three fields due to space constraints, but we use the data in the calculations for the overall microprocessor). It is clearly shown that the Reorder Buffer is the most vulnerable component among all other hardware structures for both microarchitectures. There is only one exception for the Cortex-A72, in which the L1 instruction cache is the most vulnerable one. Furthermore, Reorder Buffer is vulnerable *only* to Assert fault class (as the Load and Store Queues), in contrast to all other hardware structures that we have studied. Furthermore, as we can also see in Fig. 8, in most of cases, optimization level O0 is the most vulnerable case among all other optimization levels (O1, O2, and O3) for both microarchitectures. Reorder Buffer is the only microarchitectural component which shows less vulnerability in all optimization levels (O1, O2, and O3) compared to the unoptimized code (O0) and for both microarchitectures. For example, L1 data and instruction caches show lower vulnerability in all optimization levels than the unoptimized code, only for Cortex-A72.

V. AVF Analysis per Harware Component

In the previous section, we discussed in detail the sensitivity and behavior to all compiler optimization levels in 8 major components of our analysis and for two different microarchitectures. In this section, we quantify the aggregate effect of these vulnerabilities to the total AVF of all components (including all sub-components/fields). Different workloads and optimization levels have different AVF per component, thus in this section we utilize the per-component vulnerabilities to calculate the total AVF for every hardware structure, and in what extent each different optimization level affects it when all our benchmarks are jointly considered as the executed workload. To this end, in this section we use the weighted AVF as it is calculated in the previous section to show the comparative results and draw essential conclusions about the impact of different compiler optimization levels and microarchitectures in microprocessors vulnerability.

Fig. 9 shows the weighted average AVF difference for optimization levels O1, O2, O3, relative to O0, for each component and for each different field of the hardware structure for the Cortex-A15 (top diagram) and Cortex-A72 (bottom diagram). As we can see in the bottom graph of Fig. 9, for Cortex-A72, all the largest components of the microprocessor (L1 instruction and data caches, and L2 cache for both data and tag fields) show significantly *reduced* vulnerability in all optimization levels compared to the unoptimized code (O0). However, this is not the case for the Cortex-A15 in which we can see at the top graph, that all the largest components, except for the L1 instruction cache, show increased vulnerability. This observation suggests that the most modern microarchitectures have a lower vulnerability for optimized codes. On the contrary, for all other hardware structures, (RF, LQ, IQ, ROB), both microarchitectures provide similar vulnerability trends (SQ is an outlier because it provides opposite vulnerability trends between the two microarchitectures, however, the AVF difference of SQ is slightly different compared to the unoptimized code). Specifically, IQ and ROB (with all their fields) show significantly lower vulnerability for all optimization levels, compared to the unoptimized code (O0). This demonstrates that compiler optimizations for instruction reordering, instruction scheduling and the code size reduction can, not only improve the execution time of the workloads, but also significantly reduce the vulnerability impact of the applications on the target hardware structures. Additionally, it is clearly shown that the higher the optimization level, the less vulnerable the Reorder Buffer is, which demonstrates that the Reorder Buffer is the single microprocessor structure that *incrementally improves its vulnerability* (for any of the ROB's filed) when higher compiler optimizations are applied.

On the other hand, the Physical Register File, and Load Queue demonstrate the opposite phenomenon; i.e., all optimization levels are more vulnerable than the unoptimized code (O0) for both microarchitectures. For example, as we can see in the bottom graph of Fig. 9, when applications compiled with the highest optimization level (O3), the vulnerability of the RF can increase up to 37% more than the unoptimized code (O0). This increment means that when the microprocessor executes a fully optimized workload, it can approximately be 1.37× more vulnerable than when executing an unoptimized code.

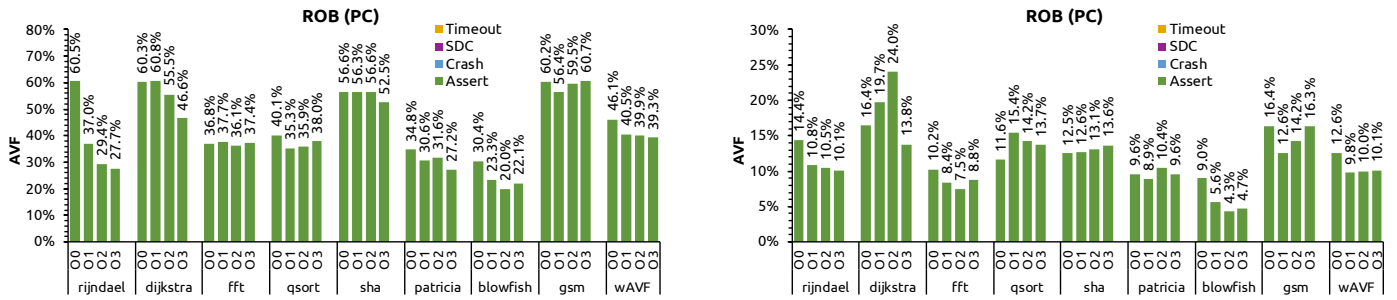

Fig. 8. AVF for Reorder Buffer for Cortex-A15 (left) and Cortex-A72 (right) for all compiler optimization levels.

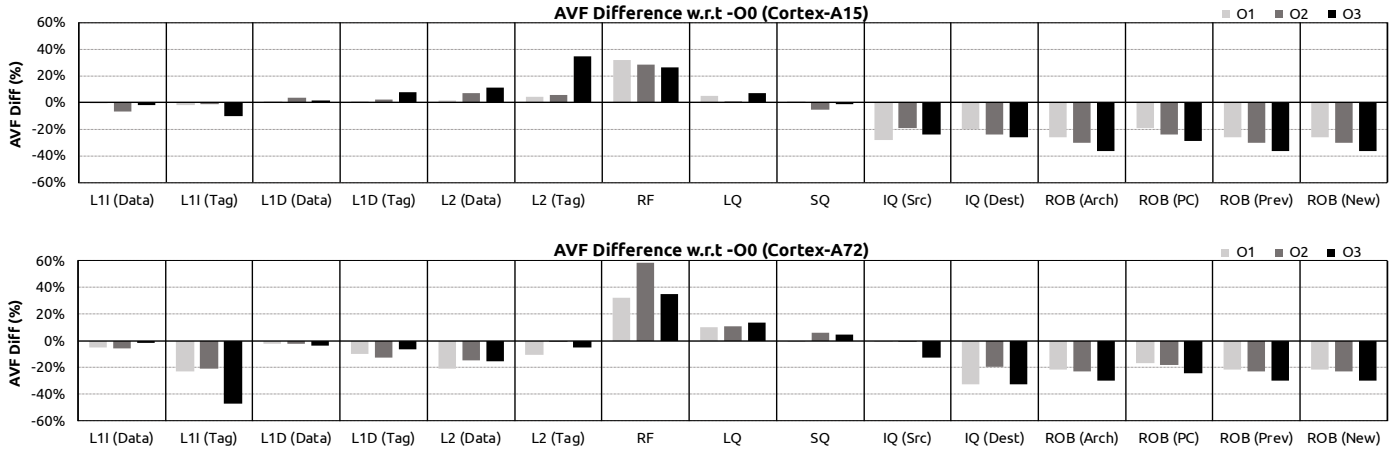

Fig. 9. Weighted AVF Difference for all optimization levels (O1, O2, O3) relative to O0, separately for each component fort both microarchitectures (Cortex-A15 at the top and Corte-A72 at the bottom).

Overall, if we compare the vulnerability difference among all optimization levels in absolute values, we can notice that the hardware component that is being affected less than all others is the Data field of the L1 data cache for both microarchitectures, while the Register File, and the Reorder Buffer are affected the most when the microprocessor executes optimized code in any optimization level.

VI. Failures in Time and Performance-Aware Comparison for the Entire Microprocessor

Apart from the vulnerability impact on each hardware structure, it is also essential to demonstrate the impact of each individual optimization level on the *entire* microprocessor chip (all hardware structures together) for each microarchitecture. To do so, in this section we first present the microprocessor's Failures in Time (FIT) rates for each individual optimization level to account for the sizes of the individual components of the microprocessor. Although the FIT rate is an essential global measurement unit of the reliability of a microprocessor, it does not take into account the application's throughput (i.e., number of tasks that can be executed per time unit). To this end, we also present another metric that takes this parameter into account and shows a performance-aware comparison among different optimization levels.

A. Failures in Time (FIT) Analysis

Failures in Time (FIT) rate of a device estimates the number of failures that can be expected in one billion (10^9) device-hours of operation. For each hardware structure of a microprocessor, a different FIT is computed using the formula in Eq. 2 below.

$$\text{FIT}_{\text{struct}} = \text{AVF}_{\text{struct}} \times \text{raw FIT}_{\text{bit}} \times \#\text{Bits}_{\text{struct}} \quad (2)$$

The FIT of the structure depends on three parameters: (1) the FIT_{bit} (or raw FIT) rate, which is determined by the fabrication technology and the operational conditions and expresses the fault probability of a single bit, (2) the number of bits of the structure, and (3) the AVF of the structure, which is affected by the microarchitecture and the executed workload. The raw FIT rate expresses the number of transient faults that will be introduced in the component, while the AVF is the derating factor that quantifies how many of these upsets will lead to a failure. The product equals to the FIT rate of a particular hardware structure. The FIT rate of the entire CPU is calculated by adding the individual FITs of the structures. For the calculation of the FIT, we use the raw FIT rate per bit, as described in [37], which is 2.59×10^{-5} FIT/bit for Cortex-A15 and 9.39×10^{-6} FIT/bit for Cortex-A72 but, of course, any technology-related (or arbitrary) value can be used.

As a first quantitative comparison, we show the FIT rates for each optimization level and for all benchmarks, as shown in Fig. 10. The most important observation is that in Cortex-A72, 5 out of 8 benchmarks (*dijkstra*, *fft*, *sha*, *blowfish*, *gsm*) have significantly lower FIT rate compared to Cortex-A15. Another important observation, is that in the most modern microarchitecture (Cortex-A72), there is a significant increase of the SDC rate. As we can see, in Cortex-A72, the SDC rate is the dominant one, while in Cortex-A15 the AppCrash is the dominant. This observation suggests that modern microarchitectures are less susceptible to SDCs, which would lead to severe problems in the field operation. This is in line with the recent works of Facebook [42] and Google [43], which unveil the significance of the

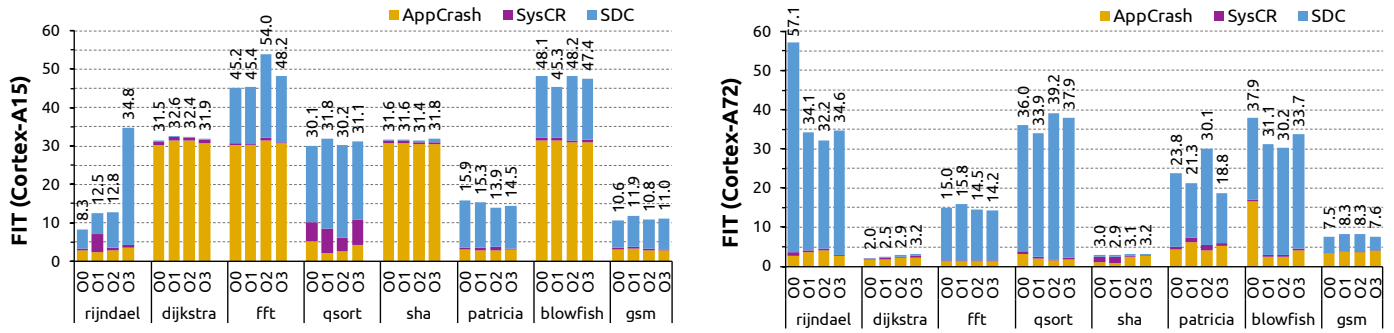

Fig. 10. Arm Cortex-A15 FIT rates (left graph) and Arm Cortex-A72 FIT rates (right graph).

SDC phenomena at scale. Moreover, as we described in Section IV, each optimization level affects differently the vulnerability of each hardware structure. Therefore, it would be misleading if we just compare the FIT rates of each optimization level. To this end, we present in the next subsection a performance-aware comparison of the failures for each optimization level to quantitatively evaluate the impact of compiler optimization levels on the microprocessor.

B. *Failures per Execution (FPE)*

Optimizations introduced by compilers significantly affect the performance of the executed code and modify the access patterns to the microprocessor's structures. To account for these diversities, system performance can be measured by the time it requires to complete a full program execution. Since our baseline out-of-order microprocessors is the same for all optimization levels, the throughput can be considered as one full execution for each microarchitecture. Assume for example in *qsort* algorithm that O0 requires 10 seconds to complete a sorting of an array (*qsort_O0*) and O2 requires 1 second for the same task (*qsort_O2*). Given that FIT rate encapsulates the failures per 10^9 device-hours, we can conclude that *qsort_O2* has been executed 10 times more than *qsort_O0*. This is not a fair comparison when we account for the same algorithm but different compilation outcomes. To this end, we introduce the *Failures Per Execution* (FPE) metric, as shown in equation 3 below:

$$\text{FPE} = \text{FIT} \times \text{Execution\_Time} / 10^9 \quad (3)$$

FPE summarizes the failure probability of each program during *a single* execution. The metric depends both on the throughput of instructions execution and on the vulnerability of the execution. For the same workload that is compiled in different ways, *lower FPE indicates better tradeoff between reliability and performance: more correct executions take place between failing executions*. Fig. 11 shows the FPE measurements normalized by the O0, for each optimization level and for all benchmarks and microarchitectures, which provides, a *fair performance-aware comparison* of the impact of optimization levels on microprocessors reliability. Using FPE we can get a more objective idea of the difference between the performance and reliability. As a tradeoff, the results in Fig. 11 show that *the performance gains offered by higher optimization levels can recover the reliability penalty (larger vulnerability) they introduce* for both microarchitectures. In Fig. 11 we can see that all benchmarks show that higher optimization levels, in general, provide lower (thus *better*) FPE w.r.t O0. For Cortex-A15, *Rijndael* and *fft*, however, show that higher optimization levels cannot cover the reliability penalty they introduce. Overall, among the optimization levels the worst tradeoff between performance and reliability is observed for O3, while O1 and O2 tend to be the best choice for the most benchmarks, especially for Cortex-A72.

VII. Discussion & Insights for Future Directions

In Section IV we presented a fine-grain AVF analysis for all major microprocessor components by providing essential observations about the impact of different compiler optimization levels on the AVF of each different application for each individual hardware structure of two different microarchitectures. However, it is hard to extract a clear trend of the overall impact of compiler optimizations on each component's vulnerability, due to the diverse effects that optimizations induce on different applications. To this end, in Section V we presented a comprehensive analysis of the impact of compiler optimizations on each individual hardware structure to thoroughly investigate the extent in which each optimization level affects each individual microprocessor component when all workloads are jointly considered. In that way, we demonstrated which components are more sensitive to higher optimization levels, and which are less sensitive. However, most modern microprocessor designs (i.e., Cortex-A72) protect L1 data cache and L2 cache with ECC protection schemes. Other designs, such as the Arm Cortex-A15, include the option of ECC protection for these hardware structures [40]. Based on this characteristic, we employ the weighted AVF shown in Section V to calculate the FIT rates of the entire microprocessor for each optimization level when all workloads are jointly considered. This way, we provide insights of the impact on different compiler optimizations on microprocessor's vulnerability when a protection scheme is either applied or not. It is clearly shown in the leftmost diagram of Fig. 12 (top line) that in Cortex-A15 implementations *without any* ECC scheme (e.g., Samsung Exynos 5250 [41]) the higher the optimization level is, the more vulnerable is the microprocessor's operation (larger FIT rates). This is not the case for Cortex-A72 (bottom line, leftmost graph) in which even in a fully unprotected design, the optimization levels are less vulnerable that the unprotected code (O0). However, when ECC is applied on both L1 data cache and L2 cache (the middle graphs) or only on L2 cache (the rightmost graphs), the O2 level is clearly the best optimization level for less vulnerable microprocessor's operation, while it delivers its performance improvements. Moreover, in most recent designs (i.e., Cortex-A72), O1 seems to be also less vulnerable, however, the O3 optimization level is the most vulnerable one for both microarchitectures when protection schemes are applied.

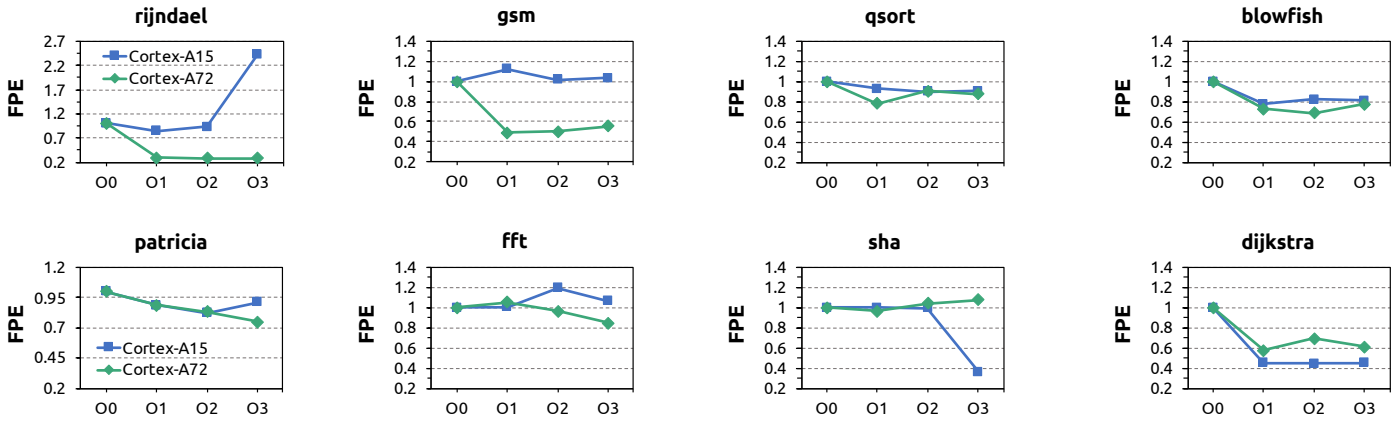

Fig. 11. Failures per Execution (FPE) normalized by the O0, for each optimization level and all benchmarks for both microarchitectures.

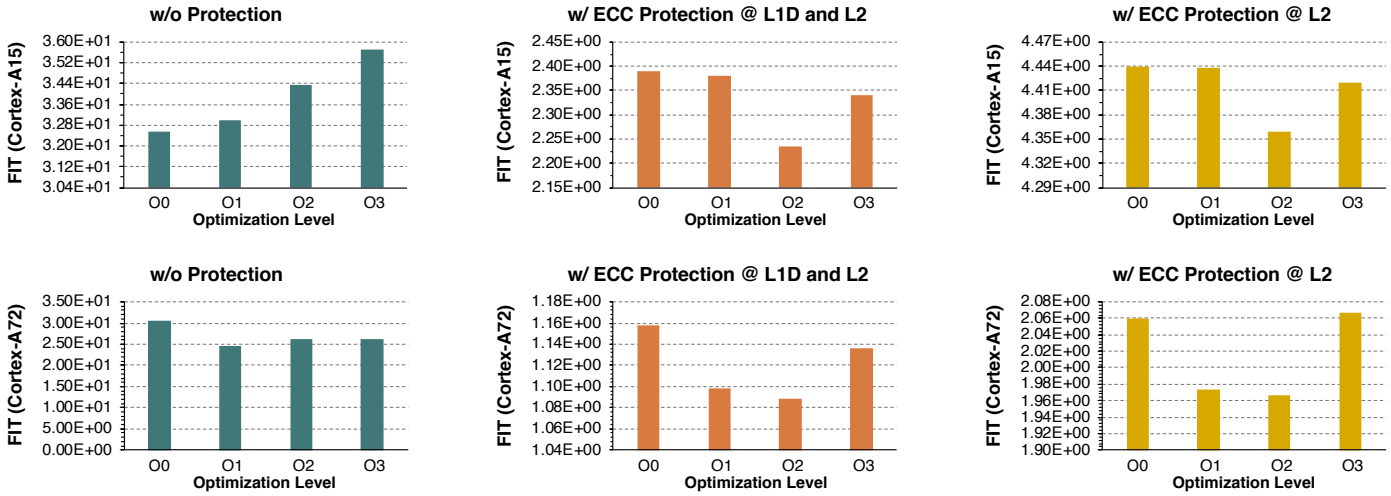

Fig. 12. Arm Cortex-A15 (top) and Arm Cortex-A72 (bottom) FIT rates for CPU designs (a) w/o ECC protection, (b) w/ ECC on L1D and L2 caches, and (c) w/ ECC only on L2.

All these insights and observations we made are essential not only for hardware architects and software designers, but also for future reliability studies. Specifically, the most common trend is that O2 and O3 optimization levels are the most frequently used choices for an application. In this study we demonstrated that in modern microprocessor designs which are equipped with protection schemes in L1 data cache and L2 cache, the O2 level is constantly the most resilient optimization, while the O3 is consistently the worst-case scenario regarding the vulnerability. However, this paper can be considered as a basis for the future reliability studies, which may consider different optimization levels depending on the scope of the research and the studied hardware structure. To the best of our knowledge, previous studies have neither provided such a fine-grained analysis nor presented any essential insight about the impact of different optimization levels on microprocessor's vulnerability.

VIII. Conclusion and Future Work

In this work, we investigated the impact of transient faults in modern out-of-order microprocessors reliability for different compiler optimization levels through extensive microarchitecture-level fault injection, which considers the effects on the entire system stack. We evaluated how different levels of compiler optimization affect the failure probability of all important hardware structures in two different out-of-order microarchitectures. We performed extensive fault injection campaigns, using large datasets for all benchmarks, to measure the Architectural Vulnerability Factor (AVF) of each optimized code, and identify that the largest microarchitectural components (L1 data and instruction caches and L2 cache – both their Tag and Data fields) for more recent microarchitecture are less vulnerable to compiler optimizations than the older ones. Moreover, we correlated the observed reliability variations to the total microprocessor's vulnerability, by showing the Failures in Time (FIT) rate for each individual optimization level and demonstrate that recent microarchitectures are less susceptible to SDCs. We also presented a performance-aware comparison (Failures per Execution – FPE) on how each optimization level affects the reliability of the microprocessor as a whole. We finally demonstrated that the O2 level is the most resilient optimization for both microarchitectures with ECC protected L1 data and L2 caches, while O1 can also offer opportunities for improved (smaller) vulnerability in the most modern microarchitectures.

As for the future work, we plan to characterize the impact of specific optimizations of each compiler optimization level on the microprocessor's vulnerability and how each optimization (or a combination of a small subset of optimizations) affects the vulnerability of each hardware structure of the microprocessor.

Acknowledgment

This research has been supported by European Union's H2020 Tetramax project (Grant 761349) through a Technology Transfer Experiment, the H2020 UniServer project (Grant 688540), and the FP7 Clereco project (Grant 611404). This work was also supported by computational time granted from the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility – ARIS.

References

[1] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, S. Di Carlo, "RT Level vs. Microarchitecture-Level Reliability Assessment: Case Study on ARM(R) Cortex(R)-A9 CPU," 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2017, pp. 117-120, doi: 10.1109/DSN-W.2017.16.

[2] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy and J. Kalamatianos, "Analysis and Characterization of Ultra Low Power Branch Predictors," 2018 IEEE 36th International Conference on Computer Design (ICCD), 2018, pp. 144-147, doi: 10.1109/ICCD.2018.00030.

[3] A. Chatzidimitriou, G. Panadimitriou, D. Gizopoulos, S. Ganapathy and J. Kalamatianos, "Assessing the Effects of Low Voltage in Branch Prediction Units," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2019, pp. 127-136, doi: 10.1109/ISPASS.2019.00020.

[4] R. Baumann, "Soft errors in advanced computer systems," in IEEE Design & Test of Computers, vol. 22, no. 3, pp. 258-266, May-June 2005, doi: 10.1109/MDT.2005.69.

[5] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu and S. Lu, "Improving cache lifetime reliability at ultra-low voltages," 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009, pp. 89-99, doi: 10.1145/1669112.1669126.

[6] 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," Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36, 2003, pp. 29-40, doi: 10.1109/MICRO.2003.1253181.

[7] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," IEEE/ACM International Symposium on Microarchitecture, 2008, pp. 129-140, doi: 10.1109/MICRO.2008.4771785.

[8] L. Yu, D. Li, S. Mittal and J. S. Vetter, "Quantitatively Modeling Application Resilience with the Data Vulnerability Factor," International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 695-706, doi: 10.1109/SC.2014.62.

[9] F. M. Lins, L. A. Tambara, F. L. Kastensmidt and P. Rech, "Register File Criticality and Compiler Optimization Effects on Embedded Microprocessor Reliability," IEEE Transactions on Nuclear Science, vol. 64, no. 8, pp. 2179-2187, Aug. 2017, doi: 10.1109/TNS.2017.2705150.

[10] M. Demertzi, M. Annavaram and M. Hall, "Analyzing the effects of compiler optimizations on application reliability," 2011 IEEE International Symposium on Workload Characterization (IISWC), 2011, pp. 184-193, doi: 10.1109/IISWC.2011.6114178.

[11] T. Jones, M. O'Boyle, and O. Ergin, "Evaluating the effects of compiler optimisations on AVF," In INTERACT '08: Proceedings of the 2008 Annual Workshop on the Interaction between Compilers and Computer Architecture in conjunction with HPCA-14, pages 112–118, 2008.

[12] N. Narayanamurthy, K. Pattabiraman and M. Ripeanu, "Finding Resilience-Friendly Compiler Optimizations Using Meta-Heuristic Search Techniques," 2016 12th European Dependable Computing Conference (EDCC), 2016, pp. 1-12, doi: 10.1109/EDCC.2016.26.

[13] B. Sangchoolie, F. Ayatolahi, R. Johansson and J. Karlsson, "A Study of the Impact of Bit-Flip Errors on Programs Compiled with Different Optimization Levels," 2014 Tenth European Dependable Computing Conference, 2014, pp. 146-157, doi: 10.1109/EDCC.2014.30.

[14] S. Bergaoui and R. Leveugle, "Impact of Software Optimization on Variable Lifetimes in a Microprocessor-Based System," 2011 Sixth IEEE International Symposium on Electronic Design, Test and Application, 2011, pp. 56-61, doi: 10.1109/DELTA.2011.20.

[15] R. R. Ferreira, R. B. Parizi, L. Carro, and A. F. Moreira, "Compiler Optimizations Impact the Reliability of the Control-Flow of Radiation-Hardened Software," Journal of Aerospace Technology and Management, vol. 5, no. 3, pp. 323–334, 2013, doi: 10.5028/jatm.v5i3.224

[16] X. Li, S. V. Adve, P. Bose and J. A. Rivers, "Online Estimation of Architectural Vulnerability Factor for Soft Errors," 2008 International Symposium on Computer Architecture, 2008, pp. 341-352, doi: 10.1109/ISCA.2008.9.

[17] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in Proceedings of the 34th ACM/IEEE International Symposium on Computer Architecture (ISCA), 2007, doi: 10.1145/1250662.1250726.

[18] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from Architectural Vulnerability," 2009 IEEE 15th International Symposium on High Performance Computer Architecture, 2009, pp. 117-128, doi: 10.1109/HPCA.2009.4798243.

[19] R. A. Ashraf, R. Gioiosa, G. Kestor and R. F. DeMara, "Exploring the Effect of Compiler Optimizations on the Reliability of HPC Applications," 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1274-1283, doi: 10.1109/IPDPSW.2017.7.

[20] C. Bender, P. Sanda, P. Kudva, R. Mata, V. Pokala, R. Haraden, and M. Schallhorn, "Soft-error resilience of the IBM POWER6 processor input/output subsystem," in IBM Journal of Research and Development, vol. 52, no. 3, pp. 285-292, May 2008, doi: 10.1147/rd.523.0285.

[21] N. Narayanamurthy, "Finding resilience-friendly compiler optimizations using meta-heuristic search techniques," Ph.D. dissertation, University of British Columbia, 2015.

[22] R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," 2009 Design, Automation & Test in Europe Conference & Exhibition, 2009, pp. 502-506, doi: 10.1109/DATE.2009.5090716.

[23] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris and D. Gizopoulos, "Differential Fault Injection on Microarchitectural Simulators," 2015 IEEE International Symposium on Workload Characterization, 2015, pp. 172-182, doi: 10.1109/IISWC.2015.28.

[24] G. Yalcin, O. S. Unsal, A. Cristal and M. Valero, "FIMSIM: A fault injection infrastructure for microarchitectural simulators," 2011 IEEE 29th International Conference on Computer Design (ICCD), 2011, pp. 431-432, doi: 10.1109/ICCD.2011.6081435.

[25] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers," ACM/IEEE 48th Annual International Symposium on Computer Architecture, 2021, pp. 902-915, doi: 10.1109/ISCA52012.2021.00075.

[26] Brian Gough, and Richard M. Stallman,, "An Introduction to GCC for the GNU Compilers gcc and g++", 2004 Network Theory Ltd.

[27] GNU Compiler Colection, Available at https://gcc.gnu.org/onlinedocs/gcc-4.7.3/gcc/Optimize-Options.html

[28] J. J. Cook and C. Zilles, "A characterization of instruction-level error derating and its implications for error detection," 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), 2008, pp. 482-491, doi: 10.1109/DSN.2008.4630119.

[29] 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," Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), 2001, pp. 3-14, doi: 10.1109/WWC.2001.990739.

[30] 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, doi: 10.1145/2024716.2024718.

[31] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016, pp. 69-78, doi: 10.1109/ISPASS.2016.7482075.

[32] 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," Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36, 2003, pp. 29-40, doi: 10.1109/MICRO.2003.1253181.

[33] N. J. George, C. R. Elks, B. W. Johnson and J. Lach, "Transient fault models and AVF estimation revisited," 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), 2010, pp. 477-486, doi: 10.1109/DSN.2010.5544276.

[34] D. S. Khudia and S. Mahlke, "Harnessing Soft Computations for Low-Budget Fault Tolerance," IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 319-330, doi: 10.1109/MICRO.2014.33.

[35] A. A. Nair, L. K. John and L. Eeckhout, "AVF Stressmark: Towards an Automated Methodology for Bounding the Worst-Case Vulnerability to Soft Errors," 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010, pp. 125-136, doi: 10.1109/MICRO.2010.34.

[36] Z. Zhao, D. Lee, A. Gerstlauer, L. K. John, "Host-compiled reliability modeling for fast estimation of architectural vulnerabilities", IEEE Silicon Errors in Logic – System Effects (SELSE), 2015.

[37] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos and P. Rech, "Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments," 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2019, pp. 26-38, doi: 10.1109/DSN.2019.00018.

[38] J. L. Henning, "Spec cpu2006 benchmark descriptions, "SIGARCH Comput. Archit. News, vol. 34, pp. 1–17, Sept. 2006, doi: 10.1145/1186736.1186737.

[39] A. Chatzidimitriou, G. Papadimitriou, C. Gavanas, G. Katsoridas and D. Gizopoulos, "Multi-Bit Upsets Vulnerability Analysis of Modern Microprocessors," 2019 IEEE International Symposium on Workload Characterization (IISWC), 2019, pp. 119-130, doi: 10.1109/IISWC47752.2019.9042036.

[40] ARM® Cortex®-A15 MPCoreTM Processor, Revision: r4p0, Technical Reference Manual, 2013.

[41] Samsung Exynos 5 dual (Exynos 5250) user's manual, Oct. 2012, https://bit.ly/3fjhR18 [Accessed online at July 8, 2021].

[42] H. Dattatraya Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar. Silent Data Corruptions at Scale. https://arxiv.org/abs/2102.11245, 2021.

[43] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," presented at the HotOS '21: Workshop on Hot Topics in Operating Systems, Jun. 2021. doi: 10.1145/3458336.3465297.