Silent Data Corruptions in Computing Systems: Early Predictions and Large-Scale Measurements

Dimitris Gizopoulos[§] George Papadimitriou[§] Odysseas Chatzopoulos[§] Nikos Karystinos[§]
Harish D. Dixit[†] Sriram Sankar[†]
[§]*University of Athens,* Greece, {dgizop | georgepap | od.chatzopoulos | n.karystinos}@di.uoa.gr
[†]*Meta Platforms Inc, Menlo Park, California*, {hdd | sriramsankar}@meta.com

***Abstract*—Silent Data Corruptions (SDCs) due to defects in computing chips (CPUs, GPUs, AI accelerators) is a critical threat to the quality of large-scale computing in different application domains: cloud computing, high-performance computing, edge computing. Recent public reports by cloud hyperscalers have emphasized that apart from the usual suspects for SDCs (memory, storage, network), the heart of the computations, the processing elements of all types generate an unexpectedly large rate of SDCs which can cause erroneous calculations and severe information loss. We report, in a consolidated form, recent efforts to correlate early microarchitecture-level simulation-based predictions about the likelihood, rates, severity, and root causes of SDCs and large-scale in-field studies in cloud data centers. Early microarchitecture-level prediction of SDC characteristics (susceptible units, workloads, instructions) can shed light to the cryptic problem of SDCs. The findings of a diligent pre-silicon analysis can assist better understanding of SDCs and can thus drive effective protection decisions either at the hardware or at the software levels at deployment stages.**

***Index Terms*—Silent data corruptions, reliability, cloud, HPC, edge, microprocessors, hardware reliability, large-scale infrastructure, microarchitectural simulation, fault injection, failure rates.**

I. Introduction

In the ever-growing large-scale computing, in which data centers serve as the backbone of modern technological infrastructures, ensuring the reliability and efficiency of hardware components is of paramount importance. One particular and severe threat in this domain is the Silent Data Corruptions (SDCs), primarily due to manufacturing defects [1]–[4]. SDCs lie in the heart of computational systems and pose a critical risk to the integrity and quality of diverse application domains, spanning cloud computing, high-performance computing, and edge computing [5]–[7]. Detecting and quantifying silent data corruptions in microprocessors is challenging due to their sporadic nature and difficult reproducibility [1], [4]. To mitigate the impact of on-chip memory errors, error correcting codes (ECC) are employed for error detection and correction [8]. However, the adoption of ECC techniques introduces additional storage demands and additional complexity and fails to address all hardware-induced errors comprehensively [9]. While conventional ECC methods can identify and rectify certain faults, their efficacy is usually constrained, with the prevalent single error correction, double error detection (SECDED) method capable of detecting up to two flipped bits and correcting only one flipped bit per 64 bits [8], [9]. Moreover, multi-bit faults are increasingly prevalent in on-chip memory structures, especially in newer fabrication technologies [10]. Despite the potential of ECC to reduce failure rates in a few on-chip memory components, its applicability is not universal across all functional, control, and memory blocks of the microprocessor chip. Even with ECC implementation, the specter of silent data corruption persists, particularly in expansive data center infrastructures, posing a substantial potential threat to program integrity [1], [4], [11].

Silent data corruptions pose a significant obstacle for modern microprocessors and the computing systems they drive. Nevertheless, researchers have made considerable strides in tackling this issue by employing advanced error detection methods and crafting fault-tolerant and error-correcting models over recent decades. As computing systems grow in complexity and importance in our daily routines, it is foreseeable that addressing silent data corruptions will remain a crucial area of focus for the computing community. It is important to pinpoint the primary sources of errors that could silently impact program execution and devise innovative strategies for modeling and detecting these errors. One approach involves implementing fault tolerance techniques, such as redundancy or replication, to ensure the availability of multiple copies of vital data. This strategy can safeguard system integrity even in the event of silent data corruption. Another avenue is the utilization of error-correcting codes capable of automatically identifying and rectifying errors. These methodologies hold particular significance in safety-critical systems, where the ramifications of silent data corruption could be catastrophic. However, driving any of these strategies requires the identification and evaluation of such errors.

Microarchitecture-level simulation offers a controlled and highly granular environment in which thousands of defective CPUs can be modeled and evaluated [12]–[14]. This approach provides researchers and engineers with a unique insight into the complicated nature of SDCs, enabling the exploration of various scenarios, probabilities, rates, severities, and root causes with high accuracy. By simulating diverse fault conditions and their potential effects, researchers can get invaluable predictive insights long before hardware deployment, facilitating proactive measures (early at the design stages) to enhance system resilience [10].

In contrast, real-world CPU fleet experiments conducted within operational data centers offer a real and concrete view of the complexities of large-scale computing environments [1]–[4]. These experiments provide invaluable empirical data regarding the prevalence, manifestations, and real-world consequences of SDCs in actual deployment scenarios. By analyzing the performance and reliability of CPU fleets under authentic or synthetic workload conditions, researchers can gain invaluable empirical insights that complement and enrich the predictive capabilities afforded by simulation-based approaches.

However, as shown in Fig. 1, despite their respective merits, it becomes increasingly evident that real CPU fleet experiments in data centers are inherently less efficient compared to microarchitecture-level simulation in certain respects. The scalability limitations, and resource constraints associated with deploying and monitoring physical hardware at scale, inevitably impose practical constraints on the scope and depth of such experiments. Consequently, while real-world experiments offer unparalleled authenticity and context, they often fall short in terms of throughput and scalability compared to their simulation counterparts.

Nevertheless, the gap between simulation-based predictions and real-world observations presents a unique opportunity for synergistic exploration and analysis. By correlating and combining insights gained from both pre-silicon microarchitecture-level simulation and post-deployment field studies, researchers can gain a holistic understanding of SDC behavior, spanning the spectrum from theoretical conjecture to empirical validation. The synthesis of findings derived from these divergent yet complementary methodologies holds the potential to drive effective protection decisions at both the hardware and software levels, thereby strengthening the resilience of large-scale computing ecosystems against the growing threat of SDCs.

In this paper, we discuss and review recent efforts aimed at bridging the gap between microarchitecture-level simulation and real-world CPU fleet experiments. A particular focus is on insights and observations about SDCs measured in different hardware units on the gem5 simulator [15]–[17]. Through a comprehensive analysis of the findings from this pre-silicon study, we aim to discuss the challenges, and opportunities inherent in the pursuit of effective SDC measurements. In such a way, we endeavor to pave the way toward a more robust, resilient, and trustworthy foundation for large-scale computing in the face of evolving threats and challenges.

The following sections are organized as follows. In Section II we discuss the concept of the silent data corruptions and why it is an important problem. In the same section, we also present the basic terminology and the main background of different types of failures, which are the focus of this paper. Next, in Section III we present an in-depth discussion about the measurement of SDCs and explore simulation-based appraches for assessing them, during both early and late design stages. In Section IV we present the methodology we follow to inject faults in (a) array-based hardware components, (b) functional hardware units, and (c) accelerator designs. Finally, in Section V we present several experimental results and provide essential observations regarding the SDC occurrences in diverse hardware structures and units. Section VI concludes the paper.

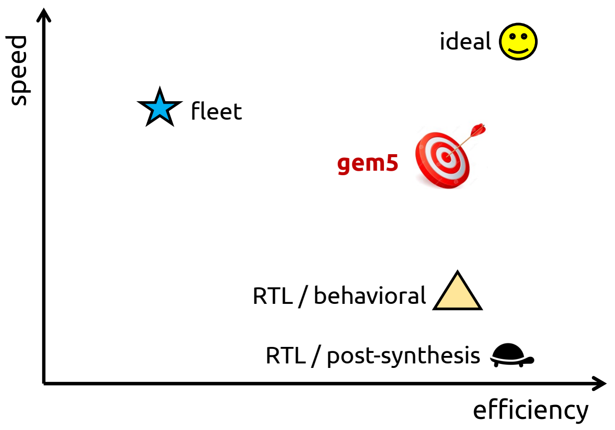

Fig. 1. A motivation graph shows the relation between speed and efficiency of correct state-of-the-art approaches for unveiling, monitoring, and measuring the rate of SDCs. An ideal method would be very fast (operating at the speed of actual systems and hardware) and efficiently identify all SDC root causes and accurate rates. Real-world (fleet) experiments are naturally very fast and run at native speed; however, they ”blindly” look for defective chips which can occasionally lead to SDCs, thus such an approach is less efficient than ideal. Simulation-based approaches at the RTL (behavioral or detailed post-synthesis) are extremely accurate and efficient in root causing since they model the real hardware; however, they are extremely slow (three or more orders of magnitude than higher-level simulation approaches). We advocate for the value of microarchitecture-level simulators like gem5 as the most balanced tradeoff point between hardware accuracy (all on-chip hardware structures are modeled) and speed of SDCs analysis (three orders of magnitude faster than RTL).

II. Background

A. Silent Data Corruptions (SDCs)

SDCs have emerged as a widespread concern, impacting critical infrastructures [1]–[3]. They are increasingly associated with CPU chips alongside memory, storage, and networking. These corruptions are labeled ’silent’ due to their evasion of hardware-level detection mechanisms, resulting in errors that may propagate through the system unnoticed until they materialize as application-level issues, potentially leading to data loss [1], [4]. SDCs can originate from various factors such as soft errors, manufacturing defects, and design flaws. They often evade detection because software does not consistently check for them, particularly in cloud and datacenter environments. Redundancy methods, such as duplicating or triplicating execution, can help mitigate SDCs, but they come with performance and power costs. Both software and hardware-based redundancy methods are costly in terms of performance, power, and design complexities.

B. Defects, Faults, Errors

Faults represent physical phenomena resulting in a deviation between an expected service and the actual service provided; in a CPU, this service entails the accurate execution of programs. Each fault originates from a specific physical cause, with processor faults commonly attributed to high-energy particles such as neutrons or alpha particles, as well as defects in silicon manufacturing or degradation over the device's lifetime. The duration of a fault's manifestation—transient, intermittent, or permanent—is intricately linked to its underlying physical cause (where "bit" refers to either a storage element or a gate output) [18]–[20]. Various *fault types*, categorized by their temporal behavior, are extensively employed. We opt for the term *fault type* over *fault model* to focus solely on the duration of fault existence, excluding their logical behavior.

In the domain of hardware faults, errors manifest as observable outcomes stemming from faults in hardware resources like storage elements or gates. Utilizing a defective resource during computation can result in Silent Data Corruption (SDC), system crashes, error notifications, or corrections through hardware resilience mechanisms such as Error Correcting Codes (ECC). Of these outcomes, SDC poses the greatest risk, potentially compromising software correctness without any discernible indication of the error [21], [22]. In the realm of Reliability, Availability, and Serviceability (RAS) design for CPUs, efforts have predominantly focused on mitigating particle-induced faults, commonly termed as "soft errors" [23]–[28]. However, recent reports from hyperscalers highlight emerging fault causes, including marginal defects capable of inducing faults under specific conditions like temperature, voltage, and workload patterns [21], [29]–[31]. These defects, coupled with device degradation in scaled technologies, pose significant concerns regarding transient, intermittent, and permanent faults over the processor's lifespan. To ensure high reliability, it is imperative to comprehend various physical root causes, their associated fault types, and the resultant errors based on the location of these faults within a processor.

C. *Microarchitecture-Level Fault Injection*

Typically, designers employ either Statistical Fault Injection (SFI) [32] or analytical methods such as the Architecturally Correct Execution (ACE) analysis [33] to obtain insights into program resilience against transient faults[1]. For transient faults, both approaches aim to measure the AVF of hardware structures, yet each has its pros and cons. SFI offers high accuracy but is slow as it necessitates multiple runs to achieve high confidence, whereas ACE analysis is swift but demands substantial development effort and may overestimate AVF (up to 3x overestimation [34]). Statistical fault injection serves as a reliability estimation technique capable of providing full-system AVF estimation by directly accessing the program output generated with injected faults. It is widely adopted for reliability assessment, offering accuracy flexibility based on the statistical sample size and providing failure samples generated through simulation. However, it entails the drawback of requiring multiple simulations, which can be time-consuming and, depending on the level of model detail, may be deemed impractical.

[1]ACE analysis is applicable solely to transient faults, whereas fault injection addresses both transient and permanent faults since it simulates the entire program execution in the presence of faults.

III. Measuring SDCs

A. *Unveiling Errors at System-Level*

Measuring SDC rates presents a significant challenge due to their silent nature, making them undetectable by hardware or software error-handling mechanisms [7]. These rates tend to be low and vary based on both faulty hardware structures and software workloads. Accurate measurement necessitates processing extensive data from numerous faulty chips, such as the billions of bytes handled in typical data center operations. Specialized equipment like hardware monitors and software-based profiling tools are essential for this task [35]. Furthermore, SDC rates are influenced by system configuration, workload, and environmental factors like harsh conditions or power fluctuations [36]–[39]. Complex applications can also elevate SDC rates. Therefore, precise measurement requires extensive experiments across diverse conditions, a process that is time-consuming, costly, and feasible primarily for owners of extreme-scale systems.

Enterprise and cloud data centers are increasingly deploying complex System-on-Chip (SoC) devices in large quantities, heightening the risk of undetected faults that can lead to unexpected crashes or silent data corruptions (SDCs). While soft errors induced by cosmic rays are well-documented [40], [41], the expansive nature of data center infrastructure necessitates consideration of SDCs stemming from manufacturing defects and reliability mechanisms in the field [1], [4], [35]. Detecting defects leading to SDCs is challenging due to the multiple conditions required for their manifestation, including specific machine instruction sequences, operating voltage, frequency, temperature conditions, and platform behaviors such as interrupts [4]. These complexities lead to limited repeatability of SDC detection tests and necessitate prolonged test durations to uncover failures, underscoring the importance of designing test methods that accommodate this behavior. One approach involves executing SDC-targeted code multiple times during tests, while another entails employing pseudo-random instruction and data sequences in each execution loop to augment the variety of data sequences applied in tests.

Numerous factors can cause faults in an SoC, such as radiation, electrical marginalities, and manufacturing defects. Even silicon defects that are not detected (or even exist) during manufacturing can result in faults [42], [43] in the field. The way these faults affect the operation of a workload depends on the circuit where the fault occurs [44], [45]. If the fault occurs in a circuit that includes error detection and correction, such as a cache or memory with an error correction code, the hardware can correct the error.

Silent data corruptions go undetected, do not interrupt the machine operation, but instead result in data errors. Data errors are more likely to occur when faults occur in circuits that are not used for program control, such as the SoC's integer of floating-point units [46]. The effects of silent data corruptions are unpredictable and depend on various factors. While an incorrect calculation of a single pixel value may not be significant, a data error in a financial transaction calculation could require corrective action [35]. Since a single fault can manifest in different ways over time due to workload variations, managing faults that can cause SDC at scale is crucial, particularly when millions of processing cores are installed in a data center or a supercomputer. Lerner *et al.* in [35] presented that a data center of modest size (i.e., 100,000 SoCs) is likely to experience at least one SDC event per month with a rate of 10 failures in time (FIT)[2]. For larger installations, frequent SDC events are likely, even at 1 FIT. To this end, it is crucial to minimize the rate of SDC, for example, by periodically testing the data center infrastructure to identify defective hardware components that perform wrong calculations [7].

B. *Exploring Simulation-Based Approaches for Assessing SDC Rates*

Given the complexities outlined in Section III-A regarding the measurement of SDC rates, it is not surprising that many researchers have turned to simulations to investigate such errors. Simulation-based analysis offers the advantage of assessing faulty chips without the need for physical access to defective hardware. However, simulations do have their limitations. Specifically, measuring SDC rates at the RTL (register-transfer level) provides exceptional detail and accuracy but comes at an exorbitant computational cost (see Fig. 1). In reality, obtaining precise SDC rates at the RTL is virtually unattainable due to the extensive time requirements, even with the most powerful contemporary computers available. Table I shows the most common ways to evaluate the reliability (including the expected SDC rates) of computing devices, comparing the time and cost required to complete the study, how many of the available resources can be accessed (or are modeled), if the faults are induced by natural processes (i.e., realistic error rates) or synthetic (i.e., models chosen by the user), if the study can be performed in the early stages of the project or only on the final product, and how much information can be gathered on faults generation and propagation (observability). Alternatively, researchers may endeavor to measure SDC rates using real machines. However, this is feasible only for hyperscalers—entities possessing extensive fleets of computing machines capable of accurately studying SDC rates [1], [4]. For instance, achieving precise SDC rate measurements may necessitate billions of machines. Despite the proliferation of cloud computing and big data platforms, acquiring access to such vast quantities of machines remains challenging. Assessing the real probability of failure (or the FIT rate) for microprocessors comprising millions of bits and programs spanning billions of cycles is extremely difficult. Notably, there are two stages at which the FIT rate is measured.

[2]1 FIT equals to one failure every 10^9 (one billion) hours of operation

Early stages: when both the design of the microprocessor hardware and the design of the software are under development, major modifications can be applied. In the early stages, the FIT rate of a microprocessor, program pair is estimated or predicted and not measured. This is because there are certain parts of the hardware structure of the microprocessor that are still unknown in detail or are deliberately removed from the abstraction to facilitate the design and simulation of the system. Analysis of the failure rates at early stages can be only implemented using architectural (ISA), or microarchitectural models of the system, both of which are available very early in the design flow. Architectural models do not include any hardware information, but only the ISA visible hardware locations (memory and registers) [47]. Microarchitectural models (also referred to as performance models) have a significant detail of the microprocessor hardware: they contain all major hardware storage components that occupy a considerable part of the final silicon estate (registers, register files, buffers, queues, caches, predictors, etc.) but they model the combinational logic and the random sequential logic (state machines in control) only functionally. Program executables (assembly/machine instructions) can run on both an architectural and a microarchitectural model. The architectural model is typically around three orders of magnitude faster to simulate than the more detailed microarchitectural one. Moreover, microarchitecture-level models can also be used for bug modeling during the CPU validation phase, e.g., [43], [48].

Late stages: when the microprocessor design, as well as the program design, are very close to completion and design changes (particularly to the hardware) are either impossible or extremely costly. At these stages, the failure rate of a microprocessor, program pair can be measured because almost all details of the hardware design are in place, unlike the early models. In the late stages, measurement of the failure rates is mainly employed for validation purposes. Late-stage measurements can be realized when the program runs on two setups: a gate-level (RT level) model of the microprocessor design, and a manufactured silicon chip. Unfortunately, the simulation speed of such fine-grained late-stage models is prohibitive to run reasonably long programs. The simulation throughput of the gate-level models is typically three or more orders of magnitude slower than the microarchitecture (performance) models. Therefore, the combined effect of the

TABLE I
SILENT DATA CORRUPTION RATE MEASUREMENT METHODOLOGIES [14].

Evaluation Method	Time Needed	Cost	Accessible Resources	Fault Source	Availability	Observability
Field, Lifetime data	months/years	very high	all	natural	final product	limited
Beam testing	hours	high	all	natural	final product	limited
Software-level fault injection	hours	low	limited	synthetic	early/final product	medium
Architecture-level fault injection	days	low	limited	synthetic	early	medium
Microarchitecture-level fault injection	**days/weeks**	**low**	**most**	**synthetic**	**early**	**very high**
RTL fault injection	years	low	all	synthetic	late	very high

hardware design and software design on the failure rate of the system cannot be measured at the gate or the RTL.

Finally, the measurement of the failure rate on actually manufactured chips is the only true physical experiment that runs at the true speed of silicon. The major drawback of this experiment is it requires expensive accelerating testing of the chips with dense beams of particles. Such particles (neutrons or others) blindly hit the chip when the program runs, and the failing executions (output corruptions or abnormal terminations) are recorded. There is no way to isolate the hardware spot that was affected and the bits that were flipped. However, the failure rates of such physical beaming experiments are employed by the industry to better emulate the actual physical conditions in an accelerated setup to reach statistically significant results for the failure rates.

Summary: Typically, RAS architects rely either on Statistical Fault Injection (SFI) [32] or on analytical methods, such as the Architecturally Correct Execution (ACE) analysis [33], to provide insights into the programs' resiliency toward transient faults because both methods aim to report the cross-layer vulnerability. Unlike lower-level simulation models (e.g., gate and RTL), microarchitecture-level fault-injection based on performance models allows deterministic end-to-end execution of large workloads on top of an operating system, i.e., full system analysis, which is impossible at lower levels [45], [47]. Further, injection on RTL models [49] would marginally augment vulnerability analyses with combinational logic vulnerability, since logic has very low raw failure rates compared to storage elements.

IV. Methodology

In this section, we provide an overview of the methodology we follow to obtain the results in Section V.

A. Array Component Fault Injection

Our microarchitectural fault injection framework [45], whether targeting array components, functional units, or accelerators, comprises three main modules.

1) The Fault Mask Generator module produces fault masks used during the injection campaign. This is a one-step process for each combination of hardware structure and benchmark. The Fault Mask Generator can generate a random set of fault masks for any type of fault (transient, intermittent, permanent) for the entire simulation time of the benchmark, based on user-defined parameters. A fault mask contains information about: (i) the processor core in which the fault will be injected (applicable in multicore architectures), (ii) the microarchitecture structure for fault injection, (iii) the exact bit (or gate) position of the injection, (iv) the specific simulation cycle or instruction when injection occurs (for transient or intermittent faults), (v) the type of fault, and finally, (vi) the population of faults (single or multiple). All generated fault masks are stored in a ”masks repository” from which the Injection Campaign Controller selects fault masks to apply.
2) Once the ”mask repository” is provided, the actual fault injection campaign can commence. The Injection Campaign Controller retrieves the masks from the repository and dispatches injection requests to the Injector Dispatcher, which communicates directly with the gem5 simulator. The interface between the Injection Campaign Controller and the individual Injection Dispatcher facilitates the transfer of user-defined parameters related to the injection (derived from the fault-mask; e.g., bit (or gate) position etc.) to the microarchitectural simulators, as well as the transmission of the results of the fault injection experiments from the microarchitectural simulator back to the Injection Campaign Controller. The final task of the Injection Campaign Controller is to archive the injection results in a ”logs repository,” which contains all log files for further processing by the Parser.
3) The final stage of the fault injection campaign involves processing the injection results and generating fault effects classification. This process utilizes a Parser, which is a configurable script responsible for categorizing faults according to their final effect. In this paper, we provide only the SDC fault effect to provide meaningful results for this category. The Parser analyzes the fault injection results and assigns them to appropriate categories based on predefined criteria.

B. Functional Unit Injection

The gem5 simulator, while providing a configurable OoO (out-of-order) engine that models all the array components of interest in sufficient detail (caches, register files, queues, etc.), offers a fundamental view of functional units, where arithmetic unit operations are implemented at a functional level, neglecting structural and intrinsic operation details. To perform Statistical Fault Injection (SFI) on functional unit logic gates and observe their impact on workload execution, we follow the following steps: (i)) generate gate-level models of functional units in C++ for seamless integration into gem5, (ii) instrument these models to enable fault injection on any gate, initially demonstrating permanent (stuck-at) faults but adaptable to other fault types, and (iii) integrate gate-level models into gem5 using a hybrid approach to minimize simulation throughput loss. These enhancements made to gem5 are directly compatible with the fault injection process detailed in the previous subsection, Section IV-A.

C. Accelerator Fault Injection

Several recent endeavors have aimed to integrate domain-specific accelerator (DSA) models into the gem5 simulation environment.Some examples are the gem5-Aladdin [50], the PARADE [51], and the addition of SystemC support [52]. Although these approaches offer either pre-RTL or RTL-based solutions, they each present notable drawbacks such as limited design space exploration, low simulation fidelity, or high design effort. To mitigate these challenges, for our fault injection purposes, we utilize gem5-SALAM [53], leveraging a dynamic graph execution engine based on LLVM. gem5-SALAM accurately models accelerator datapaths through LLVM IR analysis, provides cycle-level modeling and enables exploration of the design space while seamlessly interacting with other gem5 system modules. Architecturally, gem5-SALAM comprises a Compute Unit for datapath execution and a Communications Interface for memory access and control, facilitating efficient communication between accelerators and the host CPU. Through memory-mapped registers and DMA transactions between system memory and accelerator (accelerator memory arrays, i.e., Scratchpad Memories (SPMs) and Register Banks (RegBanks)), data transfer and computation synchronization are facilitated, enhancing the versatility and performance of accelerator designs within the gem5 environment. Similarly to functional units, the same fault injection infrastructure is utilized.

V. Results

In this section, we showcase the effects of *Transient* and *Permanent* faults on program execution. We focus on SDC outcomes, showing the probability that a fault in a specific hardware unit results in an SDC. The basic parameters of the gem5 configuration we use in this paper can be seen in Table II.

A. SDCs due to Permanent Faults in Array Components

1) L1 Instruction Cache: Fig. 2 illustrates the SDC probability outcomes for permanent faults in the L1 Instruction Cache across fifteen benchmarks of the MiBench [54] suite for the three ISAs (Arm, x86, RISC-V). As shown in Fig. 2, the SDC probability ranges from 0.1% to 2.3% for Arm ISA, 0.1% to 1.3% for x86, and 0.3% to 2.7% for RISC-V ISA. These results are expected, in the sense that a workload running with a persistent fault in any level of cache memory that stores instructions is very unlikely to survive to the end and produce a corrupted output. Faults in most fields of instruction will primarily impact the execution flow or the instruction operands, and thus, lead to a crash [5]. On average across all benchmarks, the x86 ISA demonstrates the lowest SDC probability among the ISAs studied in this paper, while the RISC-V ISA shows the highest SDC probability in most benchmarks.

2) L1 Data Cache: Fig. 3 displays the SDC probability results for permanent faults in the L1 Data Cache across the same fifteen MiBench benchmarks for the three ISAs (Arm, x86, RISC-V). As shown in Fig. 3, the SDC probability varies from 5.1% to 53.3% for Arm ISA, 4.4% to 64.7% for x86, and 4.4%

TABLE II
Major Simulator Configurations for each ISA.

Parameter	Value
ISA	RISC-V / Arm / x86
Pipeline	64-bit OoO (8-issue)
L1 Instruction Cache	32KB, 64B line, 128 sets, 4-way
L1 Data Cache	32KB, 64B line, 128 sets, 4-way
L2 Cache	1MB, 64B line, 2048 sets, 8-way
Physical Register File	128 Int; 128 FP
LQ/SQ/IQ/ROB entries	32/32/64/128

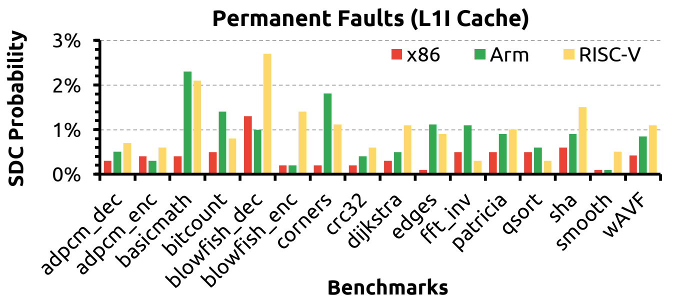

Fig. 2. SDC probability due to permanent faults in L1 instruction cache [45].

to 70.8% for RISC-V ISA. On average across all benchmarks, the RISC-V ISA exhibits the highest SDC probability among all ISAs studied in this paper. It is important to note that the L1 Data Cache is considered unprotected in our experiments, i.e., there is no ECC-related protection scheme. In real systems, the actual SDC probability can be much lower due to these protection mechanisms.

Overall, for the microarchitecture and workloads analyzed, the RISC-V ISA demonstrates a significantly higher probability of SDCs due to permanent faults compared to the other ISAs, i.e., Arm and x86.

B. SDCs due to Permanent Faults in Functional Units

As mentioned in the previous subsections, array components like L1 cache memories are typically protected by some type of error-correcting code (ECC) [8]. For example, when a cache memory employs an ECC code, (e.g., a SECDED scheme - single error correction double error detection), single-bit upsets cannot cause corruption in the data that reside in a cache memory; they are corrected by the protection scheme employed. Even in the case of a two-bit upset, the error is detected (without being corrected) and recorded in the system logs, so the resulting corruption is not silent.

On the other hand, functional units are usually unprotected by error detection and correction mechanisms. The reasoning for this choice is the following: implementing a type of ECC for functional units would increase their latency, possibly to unacceptable values. This latency is part of the processor's critical path, negatively affecting the maximum attainable clock speed, resulting in an overall slower processor design. Moreover,

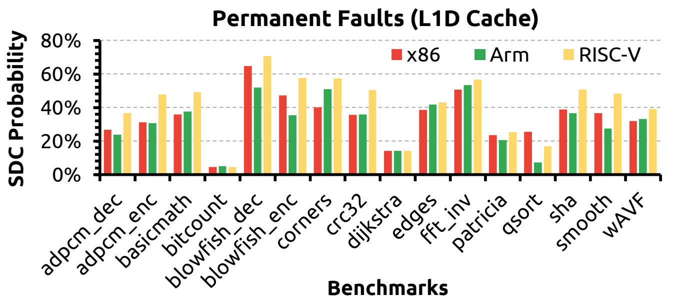

Fig. 3. SDC probability due to permanent faults in L1 data cache [45].

the complexity of the design increases as the error detection, correction, and handling mechanisms must be embedded deep in the processor pipeline (contrary to being implemented as part of the memory subsystem) where die space and power budgets are of the utmost importance.

Subsequently, functional units cannot detect or correct corruption in their computations, and thus they are good candidates for source of SDCs, posing an unmitigated risk for data corruption. In this subsection, we present SDC probabilities for 4 types of arithmetic units of a modern CPU and twenty-one workloads. The workloads are simulated on a modern microarchitectural configuration of x86 ISA, since the majority of data centers utilize x86 CPUs.

We employ a permanent gate-level fault model for these experiments on the gem5 simulator, targeting individual gates in the gate-level model of the functional units. The fault-injected gate is simulated either stuck-at-0 or stuck-at-1 for the entire program execution. Transient faults in functional units' logic are very likely to be masked. The results we present are related to the following arithmetic units: (i) integer adders, (ii) integer multipliers, (iii) vector floating-point adders, and (iv) vector floating-point multipliers.

Integer adders are of the utmost importance for any type of workload; control flow decisions (i.e., comparisons), as well as address calculations (performed by the integer adders), are present in nearly every program. Corruptions in an integer adder calculation have a substantial probability of affecting a control flow decision or memory access, which in most cases results in a program crash. This observation explains the main layout of Fig. 4. All workloads have high utilization of the integer adder, however very few exhibit substantial SDC probability. The physical characteristics of the integer adder's usage combined with the nature of permanent faults result in a high probability of crashes.

The only benchmarks exhibiting slightly higher probabilities for SDC are *sha256* (around 16%) and *opensslsha* at 20.2%. This is due to the canonical control flow of these workloads, and their heavy utilization of the integer adder for data flow, unlike the behavior observed in the rest of the workloads.

The results for the integer multiplier shown in Fig. 5 exhibit a similar trend to the results for the integer adder; however, the probabilities are increased across the board, with *FFT* and *iFFT* (inverse FFT) reaching nearly 35%. This increase is expected, due to the integer multiplier being used for data flow operations more often than the adder on average. This means that crashes, when a permanent fault occurs in the integer multiplier, are not as frequent as in the integer adder.

In this paper, we also present experimental results for the two vector floating-point components, the adder, and the multiplier. In contrast to their integer and scalar counterparts, most applications do not widely use vector floating-point components. More specifically, the gray-labeled benchmarks in the x-axes of Fig. 6 and Fig. 7 do not use the vectorized floating-point components at all. These components' workloads with extensive usage are usually numerical or linear algebra algorithms operating on floating-point data. In our experiments, the workloads with the highest probabilities for SDC in the VFADD and VFMUL components fit this description: matrix multiplication (GEMM) with 98.7% in the adder and 49.6% in the multiplier, singular value decomposition (SVD) with 61.7% in the adder and 45.6% in the multiplier, sparse linear system solving (sparse) with 97.4% in the adder and 44.8% in the multiplier and FFT and inverse FFT with approximately 30% and 25% for the adder and multiplier respectively.

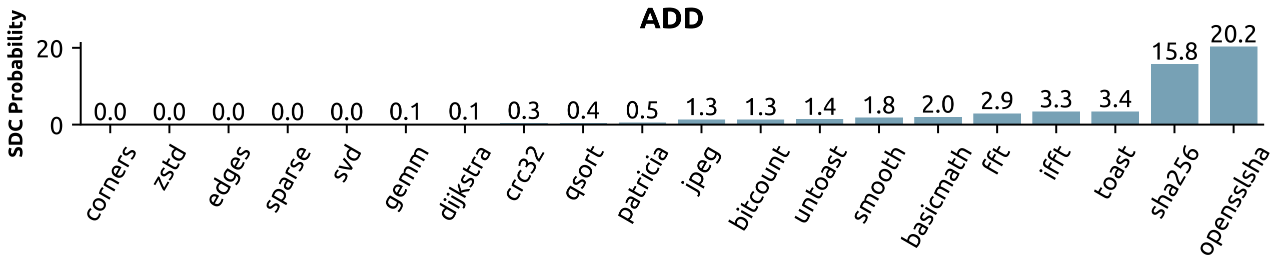

Fig. 4. SDC probability due to permanent stuck-at faults in an integer addition arithmetic unit.

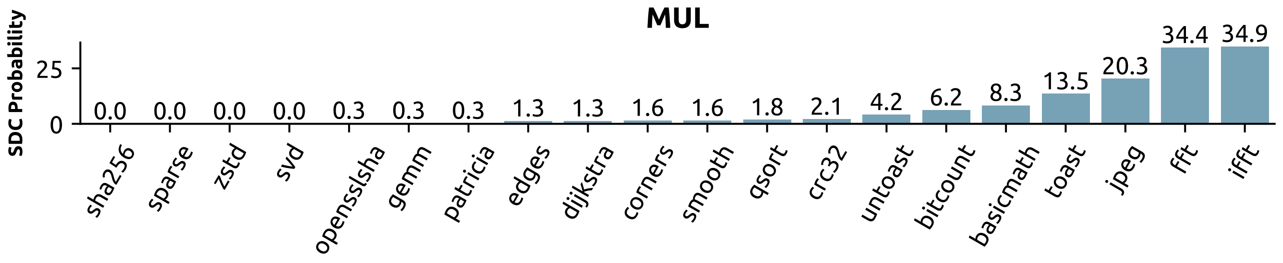

Fig. 5. SDC probability due to permanent stuck-at faults in an integer multiplication arithmetic unit.

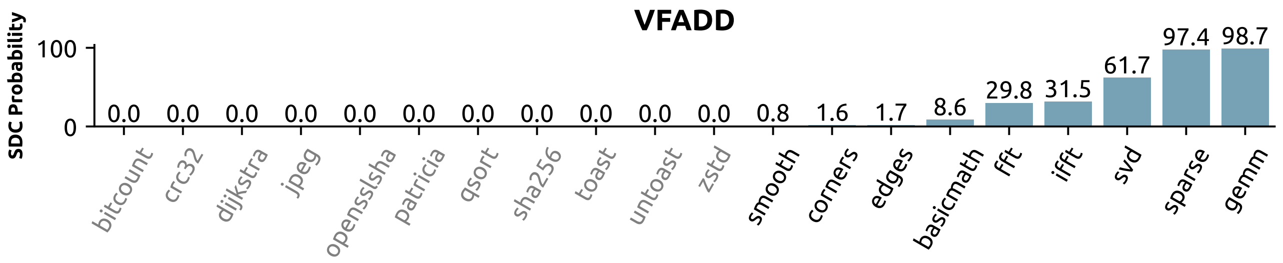

Fig. 6. SDC probability due to permanent stuck-at faults in a vector floating-point addition arithmetic unit.

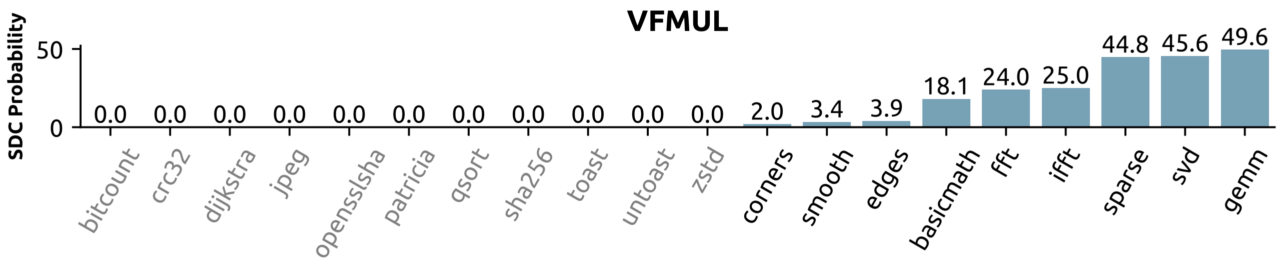

Fig. 7. SDC probability due to permanent stuck-at faults in a vector floating-point multiplication arithmetic unit.

C. SDCs due to Transient Faults in Accelerator Memory Arrays

In this subsection, we present the probability of SDC obtained from fault injection on eight distinct DSA designs, targeting their large on-chip SRAMs: scratchpad memories (SPMs) and register banks (RegBanks). These components store input, output data, and intermediate results of accelerated algorithms. For each DSA, we select representative SPMs and RegBanks for independent fault injection campaigns to assess their SDC likelihood, as shown in Table III. Fig. 8 shows these results, for all designs.

The *BFS* accelerator design employs two distinct RegBanks for accessing the EDGES and the NODES of the input graph, and does not utilize any SPMs. The SDC probability of both the EDGES RegBank and the NODES RegBank is 0%. This is because nearly all fault effects of *BFS* are Crashes. This is attributed to data from both RegBanks being utilized as indices for graph traversals by the accelerator hardware. Consequently, faults in any RegBank result in either excessively long execution times or out-of-bounds memory accesses that exceed the size of the system's physical memory. The *FFT* design employs two SPMs to store the imaginary (i.e., IMG) and REAL components of the algorithm's output. The IMG SPM has an SDC probability of 44.5%, while the REAL SPM has an SDC probability of 45.1%. These numbers are quite similar because a fault in either the imaginary or real part of the *FFT* result has an equal probability of corrupting the accelerator output. Since the SPM data is not utilized by any accelerator control logic or used as indices for memory accesses, all faults are either masked or result in SDCs. The same pattern is also observed in the *GEMM* and *MERGESORT* designs. *GEMM* stores the input data of one of the matrices to be multiplied in one SPM and the result of the matrix multiplication in another SPM, while *MERGESORT* employs two SPMs to store the main array data and temporary intermediate values. As shown in Fig. 8, the output SPM (i.e., MATRIX3) of *GEMM* exhibits a significantly lower SDC likelihood than the input SPM. This can be attributed to the injected faults in the output SPM being overwritten much more often because the input SPM gets written to only once by the DMA device when the accelerator is initialized, whereas the output SPM gets written to for the entire accelerator runtime. For *MERGESORT*, the TEMP SPM demonstrates a significantly lower SDC probability than the MAIN SPM, which can be attributed to the overwriting of

TABLE III
TARGET INJECTION COMPONENTS FOR EACH DESIGN-SPECIFIC ACCELERATOR DESIGN.

Accelerator	Component	Memory Size (Bytes)	Memory Type
BFS	EDGES	16,384	RegBank
	NODES	2,048	RegBank
FFT	IMG	8,192	SPM
	REAL	8,192	SPM
GEMM	MATRIX1	32,768	SPM
	MATRIX3	32,768	SPM
MD_KNN	NLADDR	16,384	SPM
	FORCEX	2,048	SPM
MERGESORT	MAIN	8,192	SPM
	TEMP	8,192	SPM
SPMV	VAL	13,328	SPM
	COLS	6,664	SPM
STENCIL2D	ORIG	32,768	SPM
	SOL	32,768	SPM
	FILTER	360	RegBank
STENCIL3D	ORIG	65,536	SPM
	SOL	65,536	SPM
	C_VAR	8	RegBank

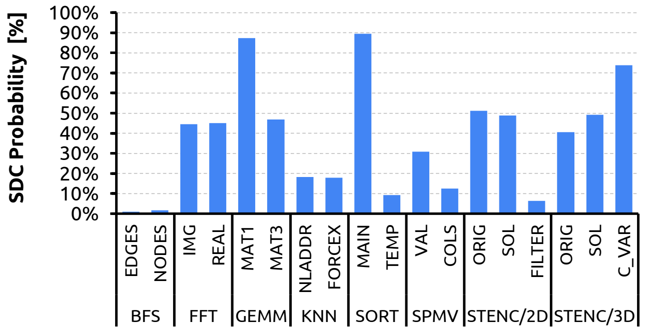

Fig. 8. SDC probability due to transient faults in accelerator memory array components [45].

numerous faults due to the continuous stream of memory writes to the SPM. Similar observations also hold for the remaining DSA designs.

VI. Conclusion

Silent Data Corruptions (SDCs) arising from defects in computing chips have emerged as a critical threat to the integrity of large-scale computing across various application domains, including cloud computing, high-performance computing, and edge computing. Recent public reports from cloud hyperscalers have highlighted the significant role of processing elements in generating SDCs, alongside traditional issues like memory, storage, and network components. In this paper, we presented a consolidated review based on recent endeavors to link early microarchitecture-level simulations with real-world observations from cloud data centers, aiming to understand the probability, rates, severity, and root causes of SDCs. Insights gained from diligent pre-silicon analysis, summarized in this paper, can shed light on the complex nature of SDCs, aiding in the development of more effective protection strategies, either at the hardware or software levels, during deployment phases.

Acknowledgment

This work was supported by Meta Platforms and AMD Research, the European Union's Horizon Europe research and innovation programme under grant agreements No 101093062 (Vitamin-V) and No 101070238 (NEUROPULS), and the School of Science of the University of Athens. Views and opinions expressed are however, those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

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

[2] H. D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, "Detecting silent data corruptions in the wild," 2022.

[3] G. Papadimitriou, D. Gizopoulos, H. D. Dixit, and S. Sankar, "Silent data corruptions: The stealthy saboteurs of digital integrity," in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2023, pp. 1–7.

[4] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores That Don't Count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 9–16. [Online]. Available: https://doi.org/10.1145/3458336.3465297

[5] G. Papadimitriou and D. Gizopoulos, "Silent data corruptions: Microarchitectural perspectives," *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3072–3085, 2023.

[6] G. Papadimitriou, D. Gizopoulos, H. D. Dixit, and S. Sankar, "Silent data corruptions: The stealthy saboteurs of digital integrity," in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2023, pp. 1–7.

[7] A. Singh, S. Chakravarty, G. Papadimitriou, and D. Gizopoulos, "Silent data errors: Sources, detection, and modeling," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–12.

[8] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[9] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 467–478.

[10] A. Chatzidimitriou, G. Papadimitriou, C. Gavanas, G. Katsoridas, and D. Gizopoulos, "Multi-bit upsets vulnerability analysis of modern microprocessors," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 119–130.

[11] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 415–426.

[12] G. Papadimitriou and D. Gizopoulos, "Anatomy of on-chip memory hardware fault effects across the layers," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 2, pp. 420–431, 2023.

[13] P. R. Bodmann, G. Papadimitriou, R. L. Rech Junior, D. Gizopoulos, and P. Rech, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *Computer*, vol. 56, no. 7, pp. 4–6, 2023.

[14] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2358–2369, 2022.

[15] 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, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[16] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[17] "gem5 GitHub Repository," https://github.com/gem5/gem5, accessed: 2023-04-25.

[18] *Digital System Testing and Testable Design.* Wiley - IEEE Press, 1990. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?bknumber=5266057

[19] D. Gizopoulos, A. Paschalis, and Y. Zorian, *Embedded Processor-Based Self-Test.* Springer, 2004. [Online]. Available: https://doi.org/10.1007/978-1-4020-2801-4

[20] D. Gil-Tomás, J. Gracia-Morán, J.-C. Baraza-Calvo, L.-J. Saiz-Adalid, and P.-J. Gil-Vicente, "Analyzing the impact of intermittent faults on microprocessors applying fault injection," *IEEE Design & Test of Computers*, vol. 29, no. 6, pp. 66–73, 2012.

[21] S. Gurumurthi, V. Sridharan, and S. Gurumurthy, "Emerging fault modes: Challenges and research opportunities," ACM SIGARCH, July 17, 2023. [Online]. Available: https://www.sigarch.org/emerging-fault-modes-challenges-and-research-opportunities/#

[22] D. Gizopoulos, G. Papadimitriou, and O. Chatzopoulos, "Estimating the failures and silent errors rates of cpus across isas and microarchitectures," in *2023 IEEE International Test Conference (ITC)*, 2023, pp. 377–382.

[23] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor," in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, 2004, pp. 264–275.

[24] D. Agiakatsikas, G. Papadimitriou, V. Karakostas, D. Gizopoulos, M. Psarakis, C. Bélanger-Champagne, and E. Blackmore, "Impact of voltage scaling on soft errors susceptibility of multicore server cpus," in *MICRO-56: 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-56. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3613424.3614304

[25] G. Papadimitriou and D. Gizopoulos, "Characterizing soft error vulnerability of cpus across compiler optimizations and microarchitectures," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 113–124.

[26] D. Sartzetakis, G. Papadimitriou, and D. Gizopoulos, "gpufi-4: A microarchitecture-level framework for assessing the cross-layer resilience of nvidia gpus," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 35–45.

[27] P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "The Impact of SoC Integration and OS Deployment on the Reliability of Arm Processors," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 223–225. [Online]. Available: https://doi.org/10.1109/ISPASS51385.2021.00040

[28] ——, "Impact of cores integration and operating system on arm processors reliability: Micro-architectural fault-injection vs beam experiments," in *2020 20th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2020, pp. 1–4.

[29] L. Peters, "New insights into ic process defectivity," Semiconductor Engineering, November 7, 2023. [Online]. Available: https://semiengineering.com/new-insights-into-ic-process-defectivity/

[30] A. Meixner, "Screening for silent data errors," Semiconductor Engineering, January 10, 2023. [Online]. Available: https://semiengineering.com/screening-for-silent-data-errors/

[31] T. C. I. S. . I. 43.040.10, "Iso/tr 9839:2023, road vehicles, application of predictive maintenance to hardware with iso 26262-5," ISO, August, 2023. [Online]. Available: https://www.iso.org/standard/83605.html

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

[33] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003, pp. 29–40.

[34] G. Papadimitriou and D. Gizopoulos, "Avgi: Microarchitecture-driven, fast and accurate vulnerability assessment," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 935–948.

[35] D. P. Lerner, B. Inkley, S. H. Sahasrabudhe, E. Hansen, L. D. R. Munoz, and A. v. de Ven, "Optimization of tests for managing silicon defects in data centers," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 578–582.

[36] A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Healthlog monitor: A flexible system-monitoring linux service," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 183–188.

[37] ——, "Healthlog monitor: Errors, symptoms and reactions consolidated," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 46–54, 2019.

[39] D. Gizopoulos, G. Papadimitriou, A. Chatzidimitriou, V. J. Reddi, B. Salami, O. S. Unsal, A. C. Kestelman, and J. Leng, "Modern Hardware Margins: CPUs, GPUs, FPGAs Recent System-Level Studies,"

[38] P. Koutsovasilis, C. D. Antonopoulos, N. Bellas, S. Lalis, G. Papadimitriou, A. Chatzidimitriou, and D. Gizopoulos, "The impact of cpu voltage margins on power-constrained execution," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 1, pp. 221–234, 2022.
in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 129–134. [Online]. Available: https://doi.org/10.1109/IOLTS.2019.8854386

[40] T. C. May and M. H. Woods, "A new physical mechanism for soft errors in dynamic memories," in *16th International Reliability Physics Symposium*, 1978, pp. 33–40.

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

[42] M. D. McCluskey and A. Janotti, "Defects in semiconductors," *Journal of Applied Physics*, vol. 127, no. 19, p. 190401, 2020. [Online]. Available: https://doi.org/10.1063/5.0012677

[43] G. Papadimitriou, D. Gizopoulos, A. Chatzidimitriou, T. Kolan, A. Koyfman, R. Morad, and V. Sokhin, "Unveiling difficult bugs in address translation caching arrays for effective post-silicon validation," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 544–551.

[44] S. Mukherjee, *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[45] O. Chatzopoulos, G. Papadimitriou, V. Karakostas, and D. Gizopoulos, "Gem5-marvel: Microarchitecture-level resilience analysis of heterogeneous soc architectures," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2024, pp. 543–559. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA57654.2024.00047

[46] I. Tsiokanos, G. Papadimitriou, D. Gizopoulos, and G. Karakonstantis, "Boosting Microprocessor Efficiency: Circuit- and Workload-Aware Assessment of Timing Errors," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 125–137. [Online]. Available: https://doi.org/10.1109/IISWC53511.2021.00022

[47] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 902–915.

[48] Y. Sazeides, A. Gerber, R. Gabor, A. Bramnik, G. Papadimitriou, D. Gizopoulos, C. Nicopoulos, G. Dimitrakopoulos, and K. Patsidis, "Idld: Instantaneous detection of leakage and duplication of identifiers used for register renaming," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 799–814.

[49] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. Kim, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, no. 2, pp. 43–52, 2005.

[50] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 97–108.

[51] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 380–387.

[52] C. Menard, J. Castrillon, M. Jung, and N. Wehn, "System simulation with gem5 and systemc: The keystone for full interoperability," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, pp. 62–69.

[53] S. Rogers, J. Slycord, M. Baharani, and H. Tabkhi, "gem5-salam: A system architecture for llvm-based accelerator modeling," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 471–482.

[54] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.