Silent Data Corruptions: The Stealthy Saboteurs of Digital Integrity

George Papadimitriou[†] Dimitris Gizopoulos[†] Harish Dattatraya Dixit[§] Sriram Sankar[§]
[†]*University of Athens,* Greece, {georgepap | dgizop}@di.uoa.gr
[§]*Meta Platforms, Inc.*, {hdd | sriramsankar}@meta.com

***Abstract*—Silent Data Corruptions (SDCs) pose a significant threat to the integrity of digital systems. These stealthy saboteurs silently corrupt data, remaining undetected by traditional error handling mechanisms. The silent nature of SDCs makes them challenging to trace at the hardware level, as they evade error reporting systems. Instead, their effects manifest at the application level, potentially causing data loss and system-wide issues. Detecting and measuring SDCs present unique challenges. Their low occurrence rates, dependence on hardware structure and software workloads, and correlation to environmental factors make accurate measurement complex. Addressing SDCs requires proactive measures to prevent data corruption and ensure digital integrity. Software redundancy methods provide a means to tolerate SDCs by introducing duplication or triplication of application resources. However, these methods come with their own limitations, including increased code size, altered execution patterns, and potential vulnerability to other types of failures. Understanding the nature of SDCs and developing effective mitigation strategies are crucial for maintaining digital integrity in large-scale infrastructure services. This paper sheds light on the stealthy saboteurs that silently corrupt data, emphasizes the need for comprehensive measurement techniques, and explores the limitations of existing mitigation approaches. By addressing the challenges posed by SDCs, we can fortify digital systems against these hidden threats and ensure the reliability and integrity of our digital infrastructure.**

***Index Terms*—Silent data corruptions, system reliability, microprocessors, hardware reliability, large scale infrastructure, microarchitectural simulation and modeling, fault injection, failure rates**

I. Introduction

In the realm of modern computing systems, microprocessors hold immense importance as critical components that drive a vast array of applications, spanning from personal devices to sprawling data centers. The future of supercomputing envisions achieving extraordinary levels of performance through the utilization of millions of microprocessor cores and specialized accelerators. However, attaining dependable computing at an exascale level remains an arduous obstacle to surmount [1], [2]. Despite employing highly aggressive design and manufacturing techniques, microprocessors are not impervious to errors. Among these, silent data corruptions (SDCs) pose a particularly insidious and hard-to-detect problem within contemporary computing [3]. Silent data corruptions occur when data becomes corrupted in a manner that appears valid, without triggering any alarms at the hardware or software level. However, the consequences manifest as incorrect results during computations. Within the realm of microprocessors, such errors can emanate from various sources, including cosmic rays, hardware defects, and design flaws. Cosmic rays, for instance, can bombard computer memory, resulting in bit flips where 0 turns into 1 or vice versa. Other sources encompass manufacturing and aging defects, design bugs, and fluctuations in power supply. Additionally, operating microprocessors at lower voltages renders them more susceptible to silent data corruptions, as the critical charge required to flip a bit diminishes at reduced voltages [4]–[11].

Detecting and quantifying silent data corruptions within microprocessors pose challenges due to their sporadic nature and difficulty in reproducing them [12]–[15]. The problem also attracted the attention of the general press [16], [17]. To mitigate the impact of on-chip memory errors, error correcting codes (ECC) are employed to identify and rectify such errors [18]. However, the use of ECC methods entails additional storage requirements and increased complexity, and it does not encompass the detection and correction of all hardware-induced errors [19]. Although commonly used ECC methods can detect and correct certain faults, their capabilities are limited. For instance, the prevalent single error correction, double error detection (SECDED) method can detect up to two flipped bits and correct only one flipped bit within a 64-bit segment [18], [19]. Furthermore, in newer fabrication technologies, on-chip memory structures witness a higher occurrence of multiple-bit faults [20]. While ECC can prove beneficial in reducing failure rates within specific on-chip memory structures, it may not be universally applicable to all functional, control, and memory blocks of a microprocessor. Consequently, even with the implementation of ECC methods, the possibility of silent data corruptions persists, particularly within extensive datacenter infrastructures, posing a significant threat to the integrity of programs [12], [13], [21].

The frequency and nature of silent data corruptions present a significant challenge for modern microprocessors and the computing systems they underpin. However, extensive research and development efforts over the past few decades have yielded remarkable progress in mitigating the occurrence and impact of silent data corruptions. As computing systems continue to advance in complexity and their importance in our daily lives grows, it is evident that this area of research will remain a focal point for the computing community. Consequently, it becomes crucial to identify the primary sources of errors that can silently compromise program execution and explore innovative methods for modeling and detecting silent data corruptions.

One approach to address this challenge is the utilization of fault tolerance techniques, such as redundancy or replication, to ensure the availability of multiple copies of critical data. This redundancy can serve as a safeguard, allowing the system to operate correctly even in the presence of silent data corruptions. Additionally, the application of error-correcting codes is another valuable strategy. These codes possess the capability to automatically identify and rectify errors, enhancing system reliability. Particularly in safety-critical systems, where the ramifications of silent data corruptions can be disastrous, such error correction methods are of utmost importance.

In this paper, we provide an overview that underscores the significance of silent data corruptions. We begin by precisely defining the nature of the problem itself. Furthermore, we comprehensively analyze existing approaches while also identifying critical gaps in addressing this pressing issue. Our investigation extends to identifying potential sources of failure that have the potential to give rise to silent data corruptions. Finally, we shed light on the challenges encountered when attempting to measure the rates of silent data corruptions in real microprocessors and detailed low-level simulation models. By employing early microarchitecture level modeling and conducting thorough measurements, we reveal the severity of silent data corruption rates across different technology fabrication nodes and under diverse operating conditions.

II. Understanding Silent Data Corruptions

A. Overview

Silent data corruption (SDC) poses a threat to large-scale infrastructure services. SDC is a widespread problem that affects microprocessor chips, as well as off-chip memory, storage, and networking, which have traditionally been identified as the main contributors to the issue [12]–[14]. SDCs are difficult to trace at the hardware level because microprocessors lack error reporting systems capable of recording such corruptions, hence the name ”silent”. However, these data corruptions permeate the system stack and manifest as problems at the application level. In large-scale data centers, the issue can be distributed across multiple server locations. The consequences of these errors include data loss, and rectifying them can take months due to the delayed detection of silent data loss [12]–[14].

Silent data corruption occurs when a microprocessor chip unintentionally corrupts the data it processes, often due to soft errors, manufacturing defects, or design flaws. For example, a CPU with a hardware fault or bug may produce incorrect computations (e.g., 2 x 2 = 5) or load/store incorrect values, which subsequently affect further computations. Unless the software actively checks for such corruptions, there may be no visible evidence of these computational issues.

Unlike other failures that are easily observable, such as application crashes, SDCs often go unnoticed in many cases. To address SDCs and prevent software-level failures, software-level redundancy methods or software-based fault-tolerant methods can be implemented in applications [22], [23]. These methods rely on redundancy, typically duplicating or triplicating the application’s resources.

B. Limitations of Software-Based Fault-Tolerance Methodologies

By doing so, fault-tolerant applications can minimize the risk of data corruption, since redundant copies increase the chances of detecting and isolating potentially malfunctioning hardware resources. However, software-based fault-tolerant methods have four significant limitations (as presented in [1]), even though they can significantly reduce the potential for SDCs and ensure correct execution.

1) The expense of incorporating redundancy in terms of performance and power efficiency is excessive, resulting in significant degradation in performance and increased power consumption. These negative effects directly impact end users. The more robust the redundancy approach, such as triplication of application resources, the greater the performance degradation and power consumption.
2) These methods aim to tolerate SDCs by introducing computational and/or resource redundancy, which significantly increases the size of the application’s code. This increase in code (and data) size alters the execution patterns of the fault-tolerant program compared to the original, unprotected version. As a consequence, while SDCs may be reduced, the likelihood of other potential faults, such as crashes, may actually increase. Recent research has demonstrated that software redundancy methods can enhance the susceptibility of hardened applications to crashes [24].
3) Software redundancy methods are typically applied only to the application itself and not to the entire software stack, including libraries and the operating system (unless they are open source). Although some previous studies have aimed to tolerate well-known open-source libraries (e.g., PyTorch [25]) and the Linux kernel (e.g., FT-Linux [26]) for hardware-induced faults, the issues of fault coverage locations and performance degradation still remain unresolved and severe. For instance, FT-Linux [26] is a Linux-based operating system that replicates race-free, multithreaded POSIX applications across different hardware partitions of a single machine. This method introduces a slowdown of up to 40% in addition to the replication, which is attributed to the utilization of double the resources. Furthermore, from both a development effort and performance degradation perspective, it would be impractical, if not unfeasible, to apply redundancy to the entire software stack, including applications, libraries, and the operating system.
4) Employing a comprehensive solution to protect the software side from hardware-induced corruptions leading to SDCs could involve hardening the entire software stack. However, recent research has indicated that a considerable number of hardware-induced faults resulting in SDCs may go undetected by both the software and architecture layers [1], [24]. Therefore, even with robust software-based protection throughout the entire

software stack, a significant portion of hardware faults that eventually cause SDCs may remain unnoticed, not only by SECDED ECC at the hardware level (especially if the corruption exceeds a single bit), but also by the software protection, and may affect the output without any indication.

C. Silent Data Corruptions at Scale

Silent Errors within hardware devices occur when an internal defect manifests in a part of the circuit which does not have checked logic to detect the incorrect circuit operation. The results of such a defect can range from flipping a single bit in a single data value, up to causing the software to execute the wrong instructions. Silent Data Corruption (SDC) can have a negative impact on large scale infrastructure services. SDCs are not captured by error reporting mechanisms or contained by hardware fault tolerance architectures within computing devices. These make them untraceable at the hardware level. However, these data corruptions propagate across the stack and manifest as application level problems. These types of errors can result in data loss and can require months of debug engineering time. Meta has observed numerous defect types in silicon manufacturing that lead to SDCs. We have dealt with hundreds of real-world examples of silent data corruption within datacenter applications and have established methodologies and debug flows to root-cause and triage faulty instructions within a computing unit. We utilize numerous techniques to implement mitigations to reduce the risk of silent data corruptions within a large production fleet. Manifestations of silent errors are accelerated by datapath variations, temperature variance, and age, among other silicon factors. Mitigations range from modifying hardware architectures for upcoming internal and external compute devices, to implementing fleetwide detection and testing architectures which scan the fleet periodically at different stages of infrastructure maintenance flows.

Given the challenging nature of the problem, we experimented with different methods for detection and mitigation at scale. We employ two such approaches:

1) Fleetscanner (out-of-production testing) and
2) Ripple (in-production testing).

We continuously evaluate the infrastructure tradeoffs associated with the silicon testing funnel across 4+ years of production experience within these techniques and consistently optimize them. In our large-scale infrastructure, we have run a vast library of silent error test scenarios across hundreds of thousands of machines in our fleet. This has resulted in hundreds of devices detected for these errors, showing that SDCs are a systemic issue across generations. Based on the at-scale monitoring established for SDCs in the past 5 years within Meta fleet, we determine that reducing silent data corruptions requires not only hardware resiliency and production detection mechanisms, but also robust fault-tolerant software architectures. The effort to mitigate these silent data corruptions require novel ideas in silicon design, verification and validation, testing strategies, hardware fault modeling and containment, compiler level instruction resilience and fault-tolerant software architectures.

III. The Challenge of Measuring SDC Rates

A. The Challenging Task of Unveiling Errors at System-Level

Measuring the rates of SDCs presents a significant challenge due to their nature - being undetectable by hardware-based or software-based error handling mechanisms. SDC rates are typically low and heavily influenced by the faulty hardware structure and the workload being executed. To obtain accurate measurements of SDC rates, a substantial volume of data from a significant number of defective chips must be processed. For instance, in a typical datacenter, billions of bytes of data are processed every second, necessitating specialized equipment and techniques like hardware monitors or software-based profiling tools [27]. Another factor complicating SDC rate measurements is the high dependency of these errors on the system's configuration and workload. Systems operating in harsh environments or experiencing frequent power fluctuations may exhibit higher SDC rates [6], [8], [9]. Similarly, systems running complex and resource-intensive applications may also encounter elevated SDC rates. Therefore, conducting accurate SDC rate measurements requires extensive experiments encompassing a wide range of conditions, which is both time-consuming and expensive. Such experiments can practically be carried out only by owners of extreme-scale systems.

Enterprise and cloud data centers are deploying increasingly intricate System-on-Chip (SoC) devices in large quantities, which heightens the risk of undetected faults that can lead to unexpected crashes or SDCs. 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 [28], [29] in the field. The way these faults affect the operation of a workload depends on the circuit where the fault occurs [30], [31]. While soft errors caused by cosmic rays are widely acknowledged [32], [33] for their ability to provide SDCs, it is essential to consider SDCs resulting from manufacturing defects and in-field reliability mechanisms due to the vast scale of data center infrastructure [12], [13], [27]. Detecting and screening defects that contribute to SDCs pose challenges because they require specific conditions for occurrence, such as a particular sequence of machine instructions, operating voltage, frequency, temperature, and platform behavior like interrupts [13]. Consequently, SDC detection tests exhibit limited repeatability, and identifying failures necessitates extended testing durations. Hence, it is crucial to design test methods that account for this behavior. One approach is to execute SDC-targeting code multiple times during tests, while another involves utilizing pseudo-random instruction and data sequences within each execution loop to increase the diversity of applied data sequences in the tests.

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 [27]. 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 [27] presented that a datacenter 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) (1 FIT equals to one failure every 10^9 –one billion– hours of operation). 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 datacenter infrastructure to identify defective hardware components that perform wrong calculations.

B. The Need for Billions of Real Machines or Billions of Years of Simulation

Given the challenges associated with measuring SDC rates, discussed in the previous section, it is not surprising that many researchers have turned to simulations as a way to study such errors: simulation-based analysis provides the opportunity to evaluate faulty chips even without having access to any faulty physical chip. Simulations have their limitations. In particular, measuring SDC rates at the RTL (register-transfer level) provides very high detail and accuracy, which, unfortunately, is extremely computationally expensive. In fact, measuring real SDC rates at the RTL is practically impossible since it can take many years, even with the most powerful computers available today. 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 processes that are natural (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 can attempt to measure SDC rates using real machines [12], [13]. However, this approach is possible only for hyperscalers, i.e., owners of huge fleets of computing machines to study SDC rates accurately. For example, in order to precisely measure the SDC rates, billions of machines may be required [12], [13]. Even with the proliferation of cloud computing and big data platforms, it is hard to obtain access to such large numbers of machines. For microprocessors consisting of several million bits and programs consisting of several billions of cycles, determining the real probability of failure (or the FIT rate) is an extremely difficult, if at all possible, task. Specifically, there are two stages at which the FIT rate is measured.

Typically, RAS architects rely either on Statistical Fault Injection (SFI) [35] or on analytical methods, such as the Architecturally Correct Execution (ACE) analysis [36], 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 [24], [31], [37]. Further, injection on RTL models [38] would marginally augment vulnerability analyses with combinational logic vulnerability, since logic has very low raw failure rates compared to storage elements.

We, therefore, employ GeFIN [5], which has been developed and extended on top of the gem5 simulator [39], which is a state-of-the-art microarchitecture-level simulator. Recent studies have shown that fault injection based on microarchitecture-level models in gem5 simulator can provide vulnerability results of the entire CPU during 18 days [31], in contrast to RTL fault injections, which could need several years (see Table I). Moreover, microarchitecture-level fault injection studies are performed in GPUs too, in which SDCs can also occur [40]. In the next subsections, we summarize a number of recent vulnerability studies using our gem5-based simulation and injection set of tools.

C. SDC Failures in Time (FIT) Analysis

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

$$FIT_{struct} = AVF_{struct} \times rawFIT_{bit} \times \#Bits_{struct}$$

The FIT of the structure is determined by three components: the FIT_{BIT} (or raw FIT) rate, which is determined by the fabrication technology and expresses the fault rate of a single bit, the number of bits of the structure and the SDC AVF of the structure, which is affected by the microarchitecture and the running workload. The raw FIT rate expresses the number of SDCs that will be introduced in the component, while the AVF (architectural vulnerability factor) is the derating factor that

TABLE I
SILENT DATA CORRUPTION RATE MEASUREMENT METHODOLOGIES [34].

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

quantifies how many of these errors will lead to a failure. The product equals the FIT rate of a component. The SDC FIT rate of the entire CPU is calculated by adding the individual SDC FITs of the individual hardware structures. In the following subsections, the AVF is determined using microarchitecture level fault injection on gem5 simulator.

Fig. 1 shows the SDC FIT rate for each technology node [20]. The red color indicates the percentage of SDC FIT due to multi-bit faults, which starts from 0% in 250 nm node and reaches a high 12% in 22 nm. We can also see that the SDC FIT for each technology node is increasing until the point of 130 nm. After that, the SDC FIT rate starts to decrease, reaching the lowest FIT values at 22 nm. These values correspond to the exact same microarchitecture with the exact same configuration. The differences observed are due to the much smaller area that the chip occupies in the higher density technologies, which results in a significantly smaller number of particles that will eventually strike the microprocessor.

D. SDC Rates for Bare-metal versus OS Executions

In this section, we summarize a characterization study which is performed through physical beam experiments on an Arm Cortex-A5 microprocessor, to show the contribution of OS (Operating System) to the SDC rates. To this end, we show a comparison between SDC FIT rates of bare-metal executions and with Linux OS. In Fig. 2 we can observe that the average SDC rates for A5 is 23.7% for bare metal and 59.3% for Linux. It is also clear from Fig. 2 that the SDC rate is constantly higher when the applications run on top of Linux, in contrast to bare-metal execution. Specifically, we can see that the difference of the SDC rates between bare-metal and Linux OS can be as high as 6.7×. However, as we discussed earlier, during beam experiments it is very difficult to investigate the SDC rates in finer granularity. To this end, in the next subsection we show results from the state-of-the-art microarchitecture-level fault injection framework, named GeFIN [41] which is based on the gem5 simulator. Simulation-based analysis (typically performed through statistical fault injection) is a very useful early-stage method for SDCs error rate estimation. Physical experiments (like the neutron beaming ones we just described) assist the validation of the simulation-based approaches and the quantification of the accuracy of the failure rates they report.

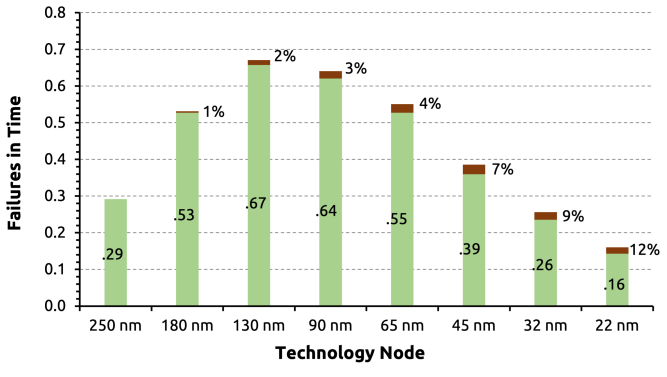

Fig. 1. SDC FIT for the entire CPU core for different technology nodes (numbers inside the green bars) due to transient faults. Red color areas correspond to the contribution of multi-bit upsets. The graph shows only the FIT rate for SDCs [20].

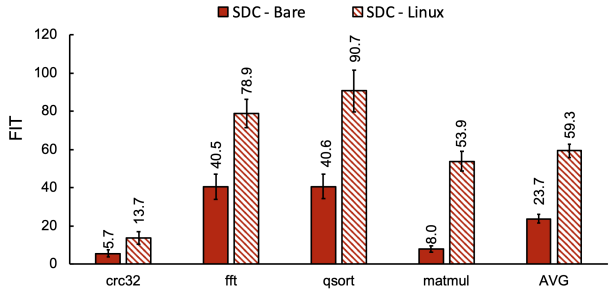

Fig. 2. Cortex A5 Bare-metal and Linux beam FIT rates for SDCs [34].

E. SDC Correlation to on-chip storage structures

In this section, we examine the relationship between SDCs and the major on-chip memory structures of modern microprocessors. It is essential to evaluate the SDC rates of individual hardware structures to understand their susceptibility. Fig. 3 illustrates the susceptibility of each structure to non-benign faults that are not masked at the hardware level. An SDC can occur if a hardware error eventually becomes available at the software and silently affects the execution of the program. Therefore, in Fig. 3 show the percentage of these errors to result in an SDC. Our first and most significant observation is that the Re-Order Buffer (ROB), Load Queue (LQ), and Store Queue (SQ) have a zero probability of experiencing SDCs. This is because any fault that occurs in these structures is not architecturally visible due to dependency graph checks that fail before the commit stage. Memory structures like the ROB, LQ, and SQ, which are deep in the microprocessor's pipeline, ensure proper instruction ordering when instructions are ready to commit. Any corruption in these structures may lead to dependency graph check failures before the commit stage and result in a crash.

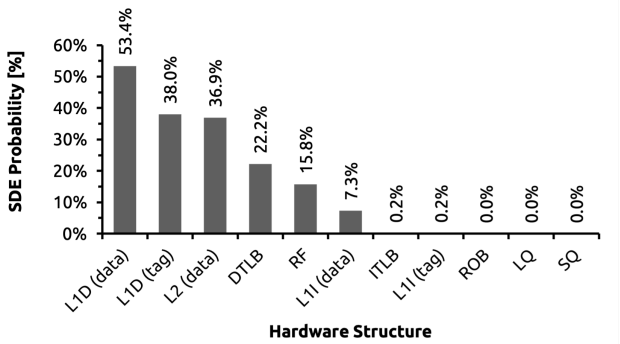

Fig. 3. The percentage of hardware corruptions at the software level (i.e., non-masked errors at hardware level) that eventually result in SDC [1].

IV. Conclusion

Silent Data Corruptions (SDCs) represent a formidable challenge to the integrity and reliability of digital systems. Their stealthy nature and ability to go undetected by traditional error-handling mechanisms make them particularly insidious. While SDCs were initially attributed to off-chip memory and storage, recent research has revealed the growing role of microprocessor chips in causing these corruptions. Measuring and detecting SDCs present significant hurdles due to their irregular rates, dependence on hardware structure and software workloads, and sensitivity to environmental factors. Accurate measurement requires specialized equipment and extensive experiments conducted under diverse conditions, making it a resource-intensive endeavor. The protection against SDCs requires a holistic approach that encompasses the entire software stack, including applications, libraries, and the operating system. However, achieving comprehensive protection remains challenging, as undetectable hardware-induced faults can still result in SDCs, evading both software-based protections and hardware-level error correction mechanisms. To ensure the integrity and reliability of large-scale infrastructure services, it is crucial to deepen our understanding of SDCs (through accurate simulation, emulation, and subsequent validation of the reported results) and, based on the above, develop effective mitigation strategies. Further research and innovation are needed to address the challenges posed by SDCs, including more robust measurement techniques, enhanced redundancy methods, and comprehensive protection mechanisms. By doing so, we can bolster the digital infrastructure and safeguard against the stealthy saboteurs that threaten the integrity of our digital systems.

Acknowledgment

Work supported by research gifts from Meta and AMD, as well as the European Union's Horizon Europe research and innovation programme under grant agreement No 101093062 (Vitamin-V), and No 101097224 (REBECCA). 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] G. Papadimitriou and D. Gizopoulos, "Silent data corruptions: Microarchitectural perspectives," *IEEE Transactions on Computers*, pp. 1–13, 2023.

[2] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 2, p. 129–173, may 2014. [Online]. Available: https://doi.org/10.1177/1094342014522573

[3] 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.

[4] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith, "Unprotected computing: A large-scale study of dram raw error rate on a supercomputer," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 645–655.

[5] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Assessing the effects of low voltage in branch prediction units," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 127–136.

[6] 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.

[7] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Analysis and characterization of ultra low power branch predictors," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 144–147.

[8] 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.

[9] 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," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 129–134.

[10] A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Healthlog monitor: Errors, symptoms and reactions consolidated," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 46–54, 2019.

[11] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, V. J. Reddi, J. Leng, B. Salami, O. S. Unsal, and A. C. Kestelman, "Exceeding conservative limits: A consolidated analysis on modern hardware margins," *IEEE Transactions on Device and Materials Reliability*, vol. 20, no. 2, pp. 341–350, 2020.

[12] 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

[13] 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

[14] H. D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, "Detecting silent data corruptions in the wild," 2022. [Online]. Available: https://arxiv.org/abs/2203.08989

[15] "Announcing the winners of the 2022 Silent Data Corruptions at Scale request for proposals," accessed: 2023-05-23. [Online]. Available: https://research.facebook.com/blog/2022/6/announcing-the-winners-of-the-2022-silent-data-corruptions-at-scale-request-for-proposals/

[16] "Tiny Chips, Big Headaches," accessed: 2023-06-06. [Online]. Available: https://www.nytimes.com/2022/02/07/technology/computer-chips-errors.html

[17] "Greek team investigating 'glitch' in computer chips," accessed: 2023-06-06. [Online]. Available: https://www.ekathimerini.com/economy/1212658/greek-team-investigating-glitch-in-computer-chips/

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

[19] 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.

[20] 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.

[21] 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.

[22] D. Kuvaiskii and C. Fetzer, "Δ-encoding: Practical encoded processing," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 13–24.

[23] M. Didehban and A. Shrivastava, "nzdc: A compiler technique for near zero silent data corruption," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[24] 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.

[25] "Pytorch elastic documentation," 2022. [Online]. Available: https://pytorch.org/elastic/0.1.0rc2/overview.html

[26] G. Losa, A. Barbalace, Y. Wen, H.-R. Chuang, B. Ravindran, and M. Sadini, "Transparent fault-tolerance using intra-machine full-software-stack replication on commodity multicore hardware," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1521–1531.

[27] 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.

[28] 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

[29] 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.

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

[31] 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. [Online]. Available: https://doi.org/10.1109/HPCA56546.2023.10071105

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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

[40] 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.

[41] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 69–78.