From Gates to SDCs: Understanding Fault Propagation Through the Compute Stack

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

***Abstract*—Silent Data Corruption (SDC) is the most severe effect of a silicon defect in a CPU or other computing chip. The arithmetic units of a CPU are, usually, unprotected and are, thus, the ones that most likely produce SDCs (as well as visible malfunctions of programs such as crashes). In this work, we shed light on the traversal of silicon defects from their point of origin deep inside arithmetic units of complex CPUs towards the program result. We employ microarchitecture-level fault injection enhanced with gate-level designs of the arithmetic units of interest. The hybrid setup combines (i) the accuracy of the hardware and fault modeling and (ii) the speed of program simulation to run long programs to end (thus observing SDC incidents); the analysis that this combination delivers is impossible at other abstraction layers which are either hardware-agnostic (software level) or extremely slow (gate-level). We quantify the effects of faults in two stages and with multiple metrics: (a) how faults propagate to the outputs of the arithmetic units when individual instructions are executed, and (b) how faults eventually affect the outcome of the program generating SDCs, crashes, or being masked. Our fine-grain findings can be utilized for informed fault detection and tolerance strategies at the hardware or the software levels.**

***Index Terms*—silent data corruption, silicon defects, microarchitectural simulation, fault injection, arithmetic units**

I. Introduction

The rapid growth of hyperscale data centers operated by companies such as Meta [1], Google [2], Alibaba [3] and others has brought to light a critical issue: an unexpectedly high incidence of Silent Data Corruptions (SDCs) in CPUs. These errors, generated by approximately one in every thousand CPUs, stem from defects in silicon and escape the error detection mechanisms at the hardware, system, or application layers. As a result, they pose a severe threat to data integrity, application correctness, compromise trust in computing [4], and incur significant engineering and operational expenses for data centers. Various factors, including CPU and system numbers, microarchitectures, manufacturing technologies, operating conditions, age, and workload profiles only increase the complexity of the problem.

Despite the focus on array units fault assessment in CPUs [5]–[15], there is a notable gap in understanding the full-system impact of faults in other critical areas, such as integer and floating-point scalar arithmetic units and their modern vector counterparts. These components are essential for computational efficiency and are tricky to protect due to their direct contribution to performance metrics like the clock cycle. Unlike arrays that can be shielded against fault using error-correcting techniques with modest performance impact, arithmetic units are more challenging to protect, and their faults often remain undetected. While transient faults in arithmetic units are less common [16], the unexpectedly high rates of SDCs reported by hyperscalers highlight a pressing need to address other types of faults in them.

Analyzing hardware problems at the microarchitecture level is the best way to balance accuracy and speed, performing broad design space exploration before the costly implementation stages. For the study of resilience, microarchitecture-level approaches enable rapid simulation — allowing fault injection experiments with high statistical significance — and support the execution of *long workloads* to completion (which is crucial to verify if the final output is compromised - thus measuring SDC incidents), all while maintaining faithful modeling of the underlying hardware. The specific requirements for such microarchitecture-level analysis depend on: (a) the hardware units of interest and (b) the fault models and their effects under consideration. Faults in any hardware structure can potentially lead to a silently corrupted output. However, the probability of SDC incidents strongly depends on (a) the operation/role of the hardware unit in instruction execution, and (b) the hardware protection scheme it encapsulates. We refer below to ”faults” in general. Still, the main focus is on faults with a more persistent nature than transients, i.e., permanent or intermittent faults, since it is commonly agreed that the SDC problem from CPUs is not due to transient faults [1], [2], [17], [18]. In particular, persistent faults in the integer ALUs of a CPU have a mixed impact on both the control flow (condition and pointer calculations) and the data flow. Thus, they have some likelihood of contributing to SDC incidents, since they are hardware units that encounter the highest frequency of use in program execution. On the contrary, other CPU components, usually hardened by a detection/correction method, are less likely to result in corruption, e.g., cache memory arrays, large buffers, and queues.

Integer arithmetic units like adders and multipliers in modern CPUs are typically unprotected, and their fault occurrence and propagation haven’t been thoroughly examined through microarchitecture-level simulations. This paper fills this gap by analyzing the impact and system-wide propagation of gate-level faults in these units. We extend the gem5 microarchitectural simulator by incorporating detailed gate-level fault injection into these critical CPU components. By injecting permanent faults (as a worst-case scenario) into gate-level models, we preserve the speed of microarchitecture-level analysis for long workloads while improving hardware modeling accuracy, unlike earlier

978-3-9826741-0-0/DATE25/© 2025 EDAA

methods that introduced significant overheads [19].

This paper addresses several key research questions to gain a more profound understanding of the critical SDC phenomenon and the way silicon defects propagate through the layers to corrupt the program output. Through these research questions, we examine the differences among various system components and explore how they behave across different workloads. Specifically, this work aims to investigate the following:

1) What is the probability that a gate-level fault propagates to the output of a functional unit, leading to errors that can potentially affect the program output?
2) What is the Bit-Error-Rate (BER) at the output of the functional unit as a result of faults?
3) How likely are crashes (i.e., visible errors) and SDCs to occur, and how frequently are faults masked or rendered inconsequential?
4) How does the BER at the outputs of the functional units correlate with the final program outcome, and what insights can we draw from this interaction?
5) In what ways does the number of error bits impact the program's behavior and its outcome?
6) What are the error rates of individual instructions due to gate-level faults, and how are these error rates correlated with the outcome of program execution?

By exploring these questions, the paper aims to provide a comprehensive understanding of how faults at the gates translate into higher-level system behavior and system reliability.

II. Background

A. Defects, Faults, Errors

A *fault* in a CPU models a physical issue (a silicon defect or an external disturbance) that causes a discrepancy between expected and actual operation, with faults manifesting as *errors*. High-energy particles cause transient faults, while silicon defects lead to intermittent or permanent faults. If a faulty hardware resource is not accessed during program execution, the fault is considered benign; otherwise, it may lead to output corruption (SDC), crash, or error notifications. SDCs are particularly problematic because they compromise data integrity and software accuracy unexpectedly [20] with any noticeable indication. Traditionally, CPU Reliability, Availability, and Serviceability (RAS) design has focused on mitigating particle-induced faults, or "soft errors" [21]. However, recent SDC reports from Meta, Google, and Alibaba suggest new fault causes, such as marginal defects that arise under specific conditions like temperature or workload patterns, leading to intermittent faults [20], [22]–[24]. Additionally, device degradation and lifetime reliability are significant in modern processors, causing both intermittent and permanent faults, necessitating an in-depth understanding of various physical fault sources and their implications [1], [2], [20].

B. Silent Data Corruptions

Silent errors within hardware devices occur when defects manifest in parts of the circuit (in our case a CPU chip), that are not equipped with any checking logic to detect incorrect results [25]. These errors can range from flipping a single bit of data to executing wrong instructions, impacting large-scale infrastructure services. SDCs escape error reporting and hardware fault tolerance, making them untraceable at the hardware level but visible as application-level issues, potentially resulting in data loss and lengthy debugging. Various defect types in silicon manufacturing contribute to SDCs, necessitating methodologies and debug flows to identify faulty instructions. Mitigation involves modifying the hardware and implementing detection/testing methodologies across the CPU fleet [26].

III. Methodology

In this section, we present in detail how we leverage the gem5 microarchitecture-level simulator [27] to simulate thousands of defective chips with faults in the gates of the functional arithmetic units of the CPU, observing their effects on actual workloads running end-to-end on top of a Linux operating system.

A. Functional unit fault injection in gem5

gem5 [27]–[29] is a versatile and widely utilized open-source simulator in the realm of computer architecture research. Its modular framework enables the simulation of various processor designs, from simple in-order cores to complex superscalar out-of-order (OoO) microarchitectures.

In this work, we employ the gem5 simulator to model a modern x86-64 microarchitecture (see Table I). While gem5 offers a configurable OoO model for simulating the specified x86 chips, its treatment of functional units is rudimentary. Arithmetic unit operations are implemented at a functional level; i.e., the unit's internal operation is not simulated, thereby ignoring all structural information and intrinsic operation details.

To conduct Statistical Fault Injection (SFI) on the logic gates of arithmetic units and observe their impact on workload execution, we undertake the following steps:

1) Generate gate-level models of functional units in C++. These models are automatically generated in C++ for easy integration into the gem5 code base. We have based our model generator on [30].
2) Instrument the C++ gate-level models to enable fault injection on any gate of the modeled functional unit [31]. For the purposes of this paper, we inject permanent (stuck-at) faults, but the models can be instrumented for *several other* fault types.
3) Integrate these gate-level models into gem5 using a smart hybrid approach aimed at *minimizing simulation*

TABLE I
MICROARCHITECTURAL CONFIGURATION.

Parameter	Value
L1 Data/Instruction Cache	32KB 8-way / 32KB 8-way
L2 Cache	512 KB 8-way
Physical Register File	192 Int; 160 FP
LQ/SQ/IQ/ROB Entries	44/48/148/256
Scalar Int FUs	5 Add; 1 Mul
Scalar FP FUs	2 Add; 2 Mul
Vector Int FUs	4 Add; 2 Mul
Vector FP FUs	2 Add; 2 Mul

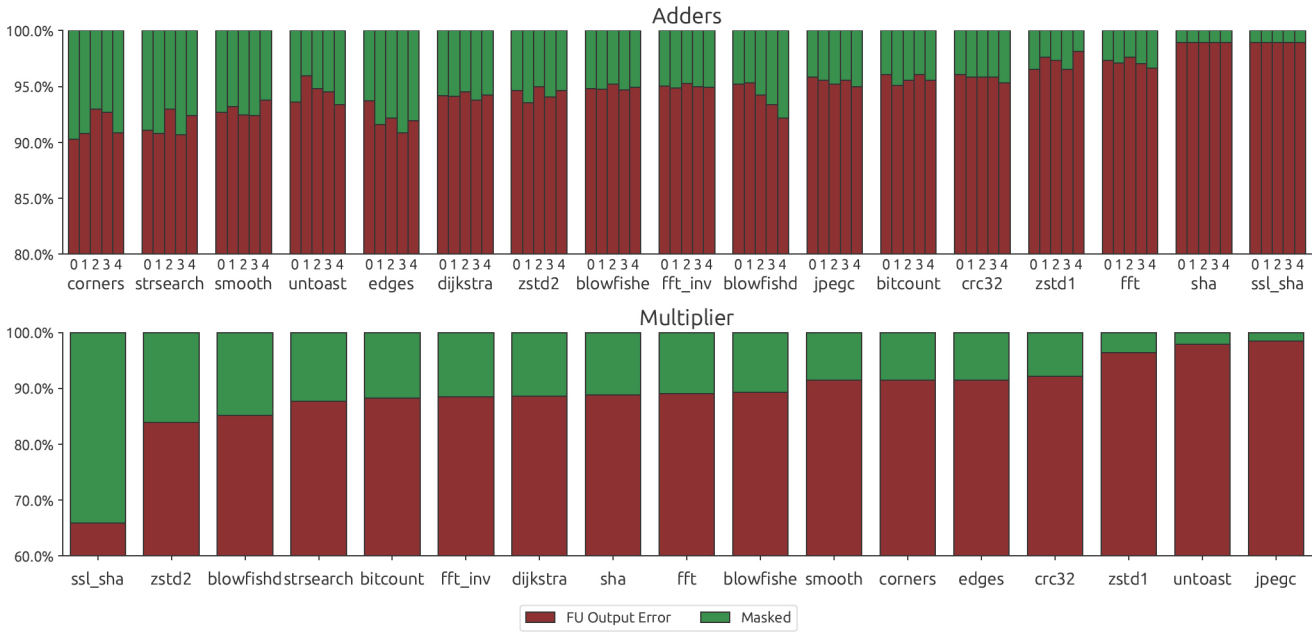

Fig. 1. Percentage of faults that propagate to the output of the functional unit for all 5 adders (top subplot) and 1 multiplier (bottom subplot) of a superscalar core across 17 workloads. Note different axis limits between adder and multiplier subplots.

throughput loss (which is less than 4 percent even for the most complex units). Older approaches that evoke an external gate-level simulator had more than 2x throughput loss [19].

4) Create a custom fault injection controller within gem5 to handle fault injection.
5) Utilize a highly optimized fault injection campaign manager designed to execute multiple parallel fault injections, leveraging the full processing speed of the host machine. Parallel fault injection and subsequent optimizations, which will be elaborated on later, significantly enhance the simulation throughput of our fault injection campaigns.

Having implemented our fault injector for arithmetic units, to obtain the results presented in the next section, we perform SFI with a sample size of 500 gates for each of the components we study. For each of the gates, we simulate a stuck-at-0 and stuck-at-1 fault scenario for a total of 1000 fault injections per arithmetic unit for each of the benchmarks we employ.

B. Benchmarks

As we highlight in the following section, the impact of faults significantly depends on the running workloads. To ensure a comprehensive analysis we employ a diverse set of 17 benchmarks. We selected 14 benchmarks from the MiBench suite [32], a collection commonly utilized in reliability studies [33], [34], due to its representative benchmarks that allow end-to-end execution in microarchitectural simulators. In addition to these benchmarks, we employ three additional benchmarks from widely used Linux libraries (included in OpenDCDiag [35]). This selection of benchmarks enables a thorough investigation of fault behavior across a broad spectrum of real-world applications and scenarios, ensuring that our findings reflect both general trends and workload-specific variations in fault sensitivity.

IV. Experimental Results

In this section, we analyze the results obtained by following the methodology described in Section III. We begin at the gate level, where we present the propagation of gate-level stuck-at faults to the outputs of functional units, highlighting the diverse behavior observed across the 17 benchmarks and two types of functional units (i.e., integer adders and multipliers[1]) (Fig. 1). To quantify the errors at the functional unit outputs, we subsequently present Bit-Error-Rate (BER) measurements for the relevant components (Fig. 2). Leveraging the simulation throughput of gem5 (around 1M simulated instructions per second), we can run the benchmarks to completion, which enables us to observe the effect of the faults on the program outputs and classify them into Crashes, Silent Data Corruptions (SDCs), or Masked (Fig. 3). By combining data on fault outcomes and functional unit output errors, we attempt to establish a relationship between the two (Figs. 4, 5). Lastly, we present instruction error rates for the Crash and SDC outcomes (Fig. 6). A broad spectrum of fault detection and recovery techniques can be built on the fine-grain findings of our analysis.

A. Functional unit gate-level fault propagation

In Fig. 1, we present the percentage of injected faults that propagate to the output of the functional units (at least once during program execution) for the five integer adders and one multiplier in the modeled microarchitecture, across the 17 benchmarks. For the adders, we observe that the benchmarks activate (cause the unit to produce at least one erroneous result) between 90% and 98% of all adder faults, while for the multiplier, they activate between 65% and 98% of faults. In the case of the adders, there is a moderate level of variability among the five adder functional units. This variation is due to the instruction scheduling mechanism of the CPU, which

[1]The adder design is a 64-bit carry-lookahead adder with 4-bit carry-lookahead blocks whereas the multiplier is a high-speed 64-bit Dadda tree multiplier.

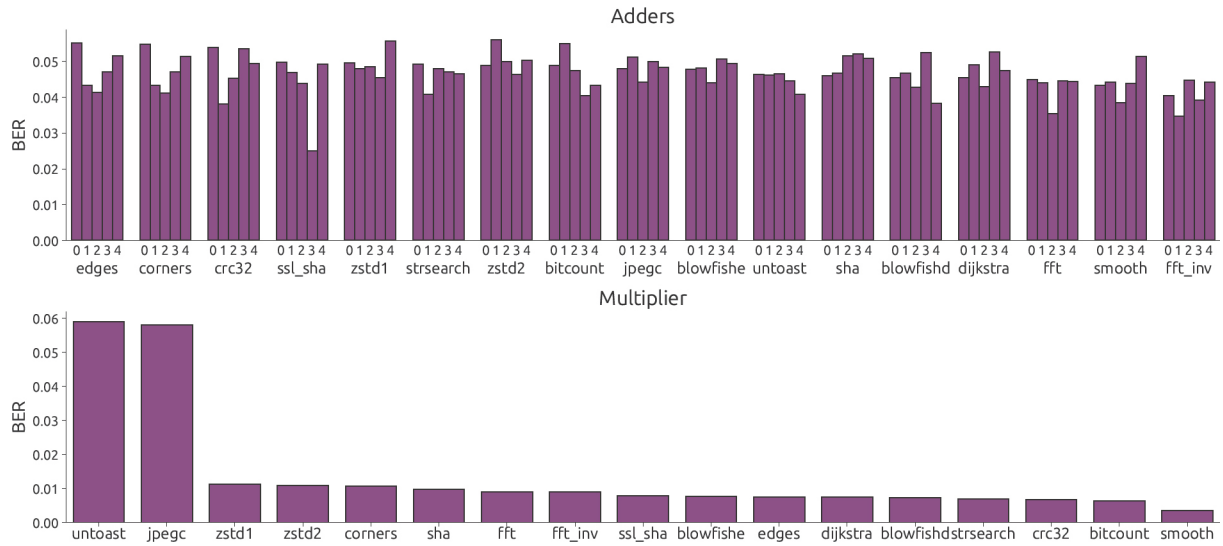

Fig. 2. Bit-Error-Rate (BER) at the output of all 5 adders (top subplot) and 1 multiplier (bottom subplot) of a superscalar core across 17 workloads.

determines which operations—and consequently, which data values—are assigned to each unit. The two implementations of the Sha algorithm exhibit the smallest degree of fault masking, likely due to the nature of the algorithm (hashing), which heavily utilizes the integer adders with diverse input data (the data to be hashed is random). In such workloads, every calculation result matters.

Observation #1: Gate-level stuck-at faults are more likely to propagate to the unit output in adders compared to multipliers. Masking strongly depends on both the workload and the scheduling, as these two factors determine the input values fed into the functional unit.

B. Functional unit Bit-Error-Rates

Fig. 2 shows the Bit-Error-Rate (BER) of each of the six functional units we analyzed. The BER is the ratio of erroneous bits over the total number of bits that were produced by the respective functional unit. For the adders, we observe similar BERs among benchmarks, ranging from 0.035 to 0.055. On the other hand, the BER of the multiplier is high for only two benchmarks (untoast, jpegc) approaching 0.06. For the rest of the benchmarks, it stays considerably lower, closer to 0.01. This can be attributed to the relatively low utilization of the multiplier by these workloads, resulting in less diverse inputs being fed into the unit.

Observation #2: For adders we observe a much more uniform BER across workloads compared to the multiplier. Adders are the most heavily utilized FUs and thus get a constant stream of inputs on almost all workloads.

C. Fault outcomes

Fig. 3 shows the fault outcomes for all injected faults in the 6 functional units across the 17 benchmarks. To obtain these outcomes, we ran the program to completion for each injected fault, totaling more than 100,000 full-system simulations for this study. A *Crash* means that either the program or the kernel crashed, preventing the program from running to completion. An *SDC (Silent Data Corruption)* occurs when the program runs to completion, but its output is corrupted. A fault is considered *Masked* when it either does not produce any errors in the functional unit's output (*Masked Internal*) or when the errors that occurred do not affect the final output of the program (*Masked External*).

For the five adders, we observe that the predominant fault outcome is Crashes. This is due to the nature of the data processed by the adders (pointers, array, and loop indices, stack addresses, etc.), making it very likely that corruption of such data will result in a Crash. Crashes for the five adders occur in over 80% of the injected faults. SDCs due to adder faults are relatively less probable, ranging from 0% to 18%. The two implementations of the Sha algorithm produce the largest number of SDCs. This can be attributed to the nature of the hashing algorithm, which makes extensive use of the integer adders, where every bit matters. Masking in the adders is relatively low, and on average, external masking (i.e., beyond the boundary of the functional unit—either at the microarchitecture or software level) is more common.

Observation #3: The predominant fault outcome for integer adders is crashes. The multiplier experiences higher masking and produces more SDCs on average.

For the multiplier (note the different y-axis ranges), we observe that Crashes are less likely, ranging from 30% to 65%. SDCs occur in several benchmarks (bitcount, fft_inv, fft, untoast, jpegc), ranging from 5% to 20%. Masking is much larger compared to the adders. This holds true for both internal and external masking. The higher rate of internal masking can be attributed to the deeper tree structure of the unit, while the increased external masking is likely due to the way software uses the multiplication data (e.g., in many cases, the upper 64 bits of a 64x64-bit multiplication are discarded).

D. Functional unit output errors and program outcomes

Fig. 4 shows the Bit-Error-Rate (BER) distribution of the three different fault outcomes (Crash, SDC, Masked) for the adders and multiplier, respectively. For the adders, we observe that Crashes, the predominant outcome, can occur across a wide

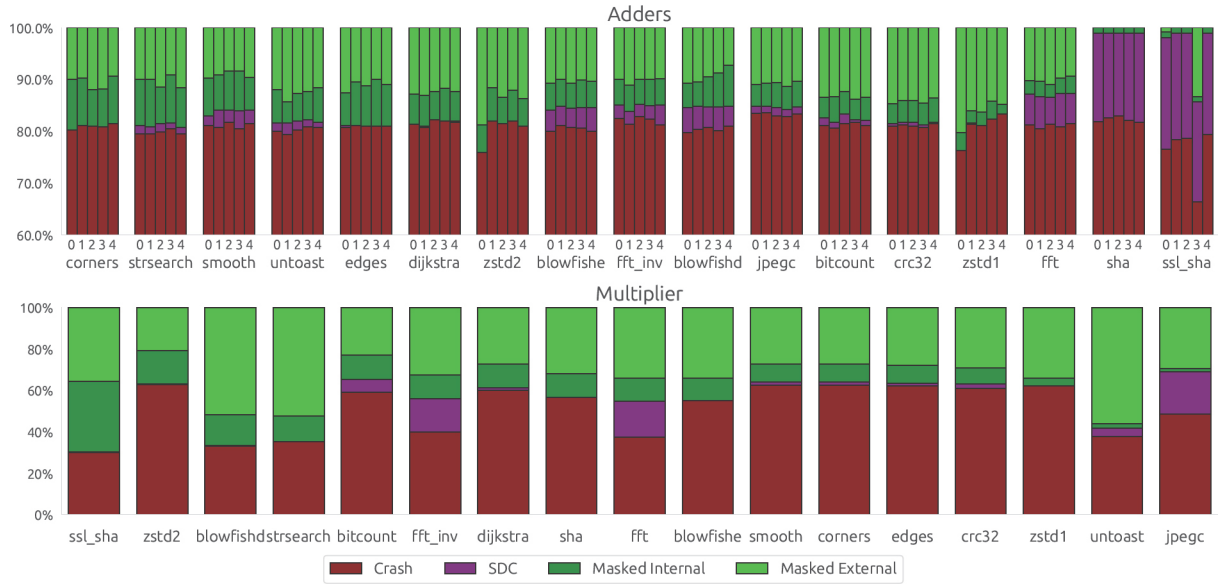

Fig. 3. Final program execution outcomes due to injected gate-level faults for all 5 adders (top subplot) and 1 multiplier (bottom subplot) of a superscalar core. Note different axis limits between adder and multiplier subplots.

range of BER values, with a relatively spread-out distribution. The mean BER for Crashes is 0.057, with a standard deviation of 0.15. The maximum BER for Crashes can be as high as 0.87. On the contrary, for the SDCs and Masked outcomes, the BER distribution is much narrower and lower, with mean BER values of 2.2×10^{-4} and 4.4×10^{-4} for SDCs and Masked outcomes, respectively. The maximum BERs for SDCs and Masked outcomes are 0.03 and 0.063, respectively.

Observation #4: For faults in adders to result in an SDC or be Masked the BER of the adder has to be extremely low. In contrast, for the multiplier, Crashes, SDCs, and Masked outcomes are likely across the BER range.

For the multiplier, SDC and Masked outcomes are much more likely, and all three distributions are considerably more spread out. This indicates a significantly lower correlation of the fault outcome with the BER for the multiplier. Both Crash and Masked outcomes have a mean BER of 0.12, with similar standard deviations. SDCs exhibit a higher mean BER of 0.05, and they even occur with a maximum BER of 0.41.

Since the same BER can result from either fewer, larger (many-bit) errors or more frequent, smaller (few bits) errors, Fig. 5 presents the fault outcomes (specifically Crashes and SDCs) as a function of the number of error bits observed at the output of the FUs. For the adders, Crashes occur across the entire range of error bits, while SDCs steadily decrease as the number of error bits increases, with a notable resurgence in the 25-52 bit range. A similar pattern is observed for the multiplier, although SDCs increase in the 17-64 error bit range.

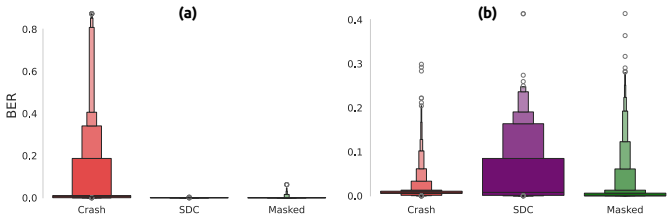

Fig. 4. Bit-Error-Rate (BER) distribution for the three possible fault injection outcomes (a) for the integer adders and (b) for the integer multiplier.

Observation #5: For adders, the occurrence of SDCs significantly decreases as the number of error bits at the unit's output increases. In contrast, for the multiplier, we observe a more uniform pattern.

E. Instruction error rates

By instrumenting the execution engine of gem5, we extract the Error Rate of instructions that pass through the 6 FUs where we perform fault injections[2]. Fig. 6 shows the measured instruction error rates for both Crash and SDC outcomes. We observe that in the case of SDCs, instruction error rates are lower on average. Additionally, several instructions are missing from the SDC subplot, indicating that these instructions (e.g., `leave`, `sysret`, etc.) almost always result in crashes. An interesting observation is that control flow and stack management instructions, such as `ret`, `call`, `push`, `pop`, etc., exhibit much lower error rates in SDC outcomes. This is because errors in these instructions are far more likely to result in crashes, as erroneous data in control flow and stack management is typically catastrophic.

Observation #6: Instruction error rates are on average higher for Crash outcomes. A lot of control flow and stack management instructions never result in SDCs in our experiments as corruptions of their values trigger segmentation faults or even kernel crashes.

[2]The error rate of an instruction is defined as the number of such instructions that use erroneous data from the arithmetic unit divided by all the instruction executions

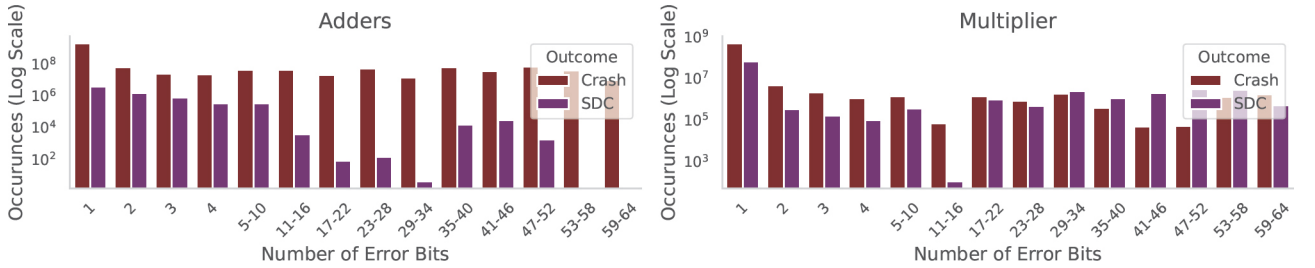

Fig. 5. Occurrences of Crashes and SDCs as a function of the number of error bits at the output of the functional unit for the adders (aggregated) (left subplot) and multiplier (right subplot). Note the log scale in the y-axis.

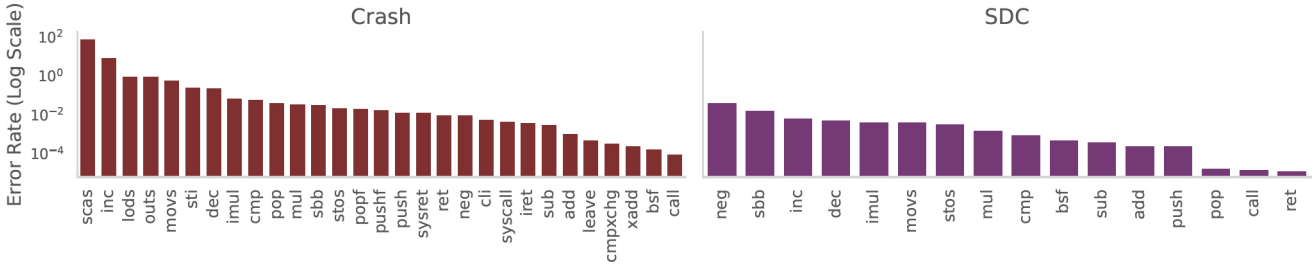

Fig. 6. Instruction error rates for Crash (left subplot) and SDC outcomes. Note the y-axis log scale.

V. Related Work

The SDC problem in computing systems has garnered significant attention recently [1], [2], [17], [25]. Previous research has extensively explored faults in CPU array structures (mostly due to transient faults) through microarchitecture-level simulation (e.g., [5], [10], [26], [33], [34], [36]–[39]). CPU array structures, nowadays, are mostly protected and are much less likely to contribute to SDCs. Our focus is on arithmetic hardware units that have virtually no protection and have been correlated with SDCs in recent reports [1]. Limited modeling of faults in arithmetic hardware units (simple ALU and address generation unit) using a microarchitectural simulator has been performed in [19]. The authors used two approaches: 1) They utilized an external gate-level simulator to model the faulty arithmetic units injecting stuck-at and delay faults. Every time the microarchitectural simulator operated on the faulty unit, the microarchitectural simulator was paused and the gate-level sim was invoked. This caused a massive overhead, reducing throughput by a factor of more than 200%, 2) They employed a statistical model at the output of the hardware units trained by gate-level simulation. While this approach was much faster, there was a significant loss in accuracy. Many studies in the literature discuss silent data corruptions, but they typically focus on the software layer and how applications are affected by these corruptions without taking the underlying hardware into account. This fails to consider the hardware-induced faults that can silently corrupt the software output [8]. Fang *et al.* in [40] presented a full-software stack study of SDCs, while Guan *et al.* in [41] conducted an empirical study of SDCs vulnerability in sorting algorithms.

VI. Conclusion

This paper investigates Silent Data Corruptions (SDCs) caused by silicon defects in CPU arithmetic units in a unique setup. Using a hybrid fault injection framework, combining gate-level fault modeling with microarchitecture simulation, we analyzed the propagation of faults through functional units and their impact on program execution. We quantified the likelihood of gate-level faults to affect the outputs, measured the Bit-Error-Rate (BER), and assessed the chances of crashes, SDCs, and masking. Additionally, we examined how BER and error bits influence the program outcomes and analyzed the error rates of individual instructions. Our findings can be employed in hardware and software protection to mitigate SDC risks, improve system reliability, and drive future research on fault tolerance and workload-specific strategies.

Acknowledgments

This work was supported by research gifts from the OCP (Open Compute Project), Meta, and AMD, as well as the European Union's Horizon Europe research and innovation programme under grant agreements No 101093062 (Vitamin-V), No 101070238 (NEUROPULS), 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. This paper is also funded by HFRI through grant No 16973 (REDESIGN).

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] 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

[3] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding silent data corruption in processors for mitigating its effects," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 4, Nov. 2024. [Online]. Available: https://doi.org/10.1145/3690825

[4] S. Hesley, "The future of computing depends on you," Keynote at the International Test Conference (ITC), San Diego, CA, USA, 2024, accessed January 17, 2025. [Online]. Available: https://www.itctestweek.org/2024-keynote-visionary-talks/

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

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

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

[8] ——, "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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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

[14] L. Yang, G. Papadimitriou, D. Sartzetakis, A. Jog, E. Smirni, and D. Gizopoulos, "Probing weaknesses in gpu reliability assessment: A cross-layer approach," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024, pp. 331–333.

[15] ——, "Gpu reliability assessment: Insights across the abstraction layers," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, 2024, pp. 1–13.

[16] 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.

[17] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding silent data corruptions in a large production cpu population," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 216–230. [Online]. Available: https://doi.org/10.1145/3600006.3613149

[18] 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.

[19] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. S. Hari, and S. V. Adve, "Accurate microarchitecture-level fault modeling for studying hardware faults," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 2009, pp. 105–116.

[20] 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/#

[21] 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.

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

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

[24] 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

[25] D. Gizopoulos, "Sdcs: A b c," https://www.sigarch.org/sdcs-a-b-c/, accessed: 2024-09-19.

[26] 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.

[27] 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

[28] 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

[29] "gem5 GitHub Repository," https://github.com/gem5/gem5.

[30] J. Klhufek and V. Mrazek, "Arithsgen: Arithmetics circuit generator for hw accelerators," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS '22)*, 2022, p. 4.

[31] O. Chatzopoulos, N. Karystinos, G. Papadimitriou, D. Gizopoulos, H. D. Dixit, and S. Sankar, "Veritas – demystifying silent data corruptions: µarch-level modeling and fleet data of modern x86 cpus," in *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, March 2025.

[32] 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.

[33] 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

[34] 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.

[35] "Intel, OpenDCDiag," https://github.com/opendcdiag/opendcdiag.

[36] 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.

[37] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, and D. Gizopoulos, "Accelerated microarchitectural fault injection-based reliability assessment," in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2015, pp. 47–52.

[38] T. Macieira, S. Gurumurthy, S. Gurumurthi, A. Haggag, G. Papadimitriou, and D. Gizopoulos, "Silent data corruptions in computing: Understand and quantify," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–7.

[39] D. Gizopoulos, G. Papadimitriou, O. Chatzopoulos, N. Karystinos, H. D. Dixit, and S. Sankar, "Silent data corruptions in computing systems: Early predictions and large-scale measurements," in *2024 IEEE European Test Symposium (ETS)*, 2024, pp. 1–10.

[40] Q. Guan, N. DeBardeleben, S. Blanchard, and S. Fu, "Empirical studies of the soft error susceptibility of sorting algorithms to statistical fault injection," in *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, ser. FTXS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 35–40. [Online]. Available: https://doi.org/10.1145/2751504.2751512

[41] Z. Li, H. Menon, D. Maljovec, Y. Livnat, S. Liu, K. Mohror, P.-T. Bremer, and V. Pascucci, "Spotsdc: Revealing the silent data corruption propagation in high-performance computing systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 10, pp. 3938–3952, 2021.