Silent Data Corruptions in Computing: Understand and Quantify

Thiago Macieira[†] Sankar Gurumurthy[*] Sudhanva Gurumurthi[*]
Amr Haggag[§] George Papadimitriou[‡] Dimitris Gizopoulos[‡]

[†]*Intel Corp.,* Portland, Oregon, USA, thiago.macieira@intel.com
[*]*AMD, Inc, Austin, Texas, USA*, {sankaranarayanan.gurumurthy | sudhanva.gurumurthi}@amd.com
[§]*Arm Ltd.,* San Fransisco, California, USA, amr.haggag@arm.com
[‡]*University of Athens,* Greece, {georgepap | dgizop}@di.uoa.gr

***Abstract*—Ensuring the reliability of hardware components is essential, particularly in large-scale installations that demand computing capabilities (cloud data centers, supercomputers, edge systems). Silent data corruptions (SDCs) due to latent defects, manufacturing testing escapes, aging, marginal defects present significant challenges to system integrity and performance. Such silicon defects are often subtle and intermittent, manifesting under specific conditions, thus complicating detection and diagnosis. In this paper, the views of the CPU design powerhouses (Intel, AMD, Arm) that deliver most of the computing power of our days at scale, are summarized. These industry giants share their insights on the challenges posed by SDCs and discuss the strategies and technologies they employ to mitigate these issues.**

***Index Terms*—Silent data corruptions (SDCs), CPUs, reliability, marginal defects, cloud computing, high-performance computing, edge computing**

I. Introduction

Ensuring the reliability and durability of hardware components is crucial, particularly in data centers and other high-demand environments. Hardware issues, notably silent data corruptions (SDCs) and small delay faults, present significant challenges to maintaining system integrity and performance. These problems are often subtle, intermittent, and can arise under specific operational conditions, making them difficult to detect and diagnose. Silent Data Corruptions are a category of error that occur when a computing system produces an incorrect result to some operation, but fails to detect it at the point where it happened. It distinguishes from errors the computing device detects and corrects (a Detected Correctable Error – DCE) and from those that could not be corrected, leading to a User Observable Error (UOE). Fortunately, SDCs are a rare enough event that most hardware will never experience it. Unfortunately, large data centers and cloud providers may possess hundreds of thousands to millions of units and, with such a large enough scale, those events will happen [1] [2].

SDCs pose a substantial challenge for modern CPUs and the computing systems they support [3]–[10]. For example, as shown in Fig. 1 a CPU with a hardware fault or bug may produce incorrect computations (e.g., 3 + 4 = 11) or load/store incorrect values, which subsequently affect further computations. Unless the software actively checks for such corruptions,

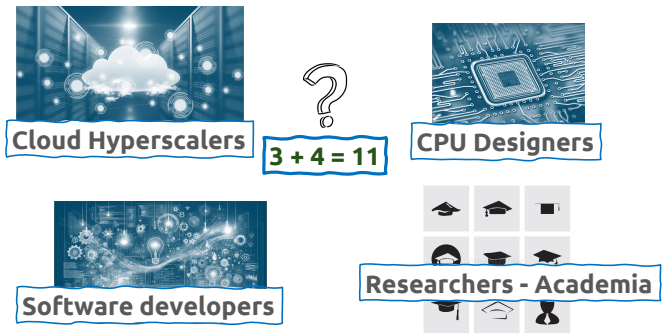

Fig. 1. Cloud hyperscalers, CPU designers, software developers, and researchers in academia put effort into identifying the primary sources of errors that can silently affect program execution.

there may be no visible evidence of these computational issues. Nevertheless, researchers have made significant efforts over the past decades in reducing the rate and impact of these errors through advanced error detection techniques [11], [12] and the development generators targeted to these errors [13]. As computing systems grow more complex and integral to daily lives, this research area will likely remain a key focus for the computing community [14], [15]. It is essential to identify the primary sources of errors (since they are not necessarily caused by soft errors [16]–[18]) that can silently affect program execution and to develop innovative methods to model and detect SDCs [19], [20].

The CPU industry has proactively addressed these challenges by developing a range of tools aimed at identifying and mitigating hardware errors during both manufacturing and field operations. A prominent tool in this suite is the Intel® Data Center Diagnostics Tool (DCDiag) [21], which is publicly available and based on the principles of an open-source project called OpenDCDiag. This paper examines the functionality and implementation of OpenDCDiag, focusing on its capabilities to detect and diagnose hardware defects through advanced software-based tests. OpenDCDiag functions by placing hardware components under significant load and employing various techniques to detect erroneous outputs. These techniques include time diversity, topology diversity, and implementation diversity, which together enhance the

effectiveness of defect detection. The framework also prioritizes test isolation, survivability, topology enumeration, and workload distribution to ensure thorough and repeatable testing conditions. Another tool in this domain is Open Field Health Check (OFHC) [22].

The paper also discusses the challenges of detecting small delay faults, which cause minor increases in signal delay that can lead to data corruption. These faults are particularly challenging to detect through traditional manufacturing tests due to their parametric sensitivity and stable conditions during testing. Online testing, both in-production and out-of-production, offers additional opportunities to detect these defects by recreating real-world conditions. The integration of design for test (DFT) techniques, functional and structural testing, and resilient design (RAS) strategies are emphasized as crucial for addressing small delay faults. The paper concludes by highlighting the importance of combining testing and RAS to ensure reliability and resilience in data center operations. Additionally, the role of on-chip sensors and a scalable sensor framework in predictive maintenance is discussed, underscoring the need for continuous monitoring and adaptation throughout the device's life cycle.

II. CPU Designer Perspective (Intel): Using OpenDCDiag to Search for Defects

Thiago Macieira

Intel has developed numerous tools to detect this type of error at manufacturing and on-site in customer premises, and has made a tool called Intel® Data Center Diagnostics Tool (DCDiag) [21] publicly available. The core of that tool is an open-source project called OpenDCDiag [23], reviewed below.

A. Overview

The two crucial aspects of a software-based hardware test are to trigger a hardware defect and to then detect its erroneous output. Triggering it usually involves placing the hardware under significant load or by using it in unconventional ways. Detecting the error implies performing either a comparison of the output data against a known-good answer or further processing it in a way that will reveal it to be incorrect.

To achieve this, software tests will usually create random input data to operate on. In contrast, testing of the software and analysis of the hardware usually require repeatable conditions, limiting the ability to use truly random data. So, it is necessary to use a deterministic, pseudo-random number generation that is in turn controllable in a way to recreate the state it had at a known point prior to the defect having been sighted.

Verifying the computations on random input data is another challenge. Software will usually avail itself of one or more of these three strategies to achieve this:

1) Time diversity: repeat the same operation over time and compare the results against previous executions.
2) Topology diversity: execute the same operation on different compute units such as CPU cores and cross-check the results.
3) Implementation diversity: execute the same logical operation using different implementation techniques and cross-check the results.

OpenDCDiag provides a framework for software developers to implement both time diversity and topology diversity without boilerplate code, freeing up time to invest in the content itself. As the OpenDCDiag framework implements the execution each of those steps in different CPU cores, this achieves both time and topology diversity.

Finally, the framework must also provide the means to produce logs and package the software for distribution.

B. Test isolation and survivability

Isolating each test from all others allows test developers to not have to worry about side effects produced by earlier tests that may have been run.

This is implemented by starting child processes for each test. On Unix systems, OpenDCDiag uses the *fork()* system call to clone itself, which is implemented on modern OSes by marking all writable memory pages as *copy-on-write*. Thus, any changes to memory caused by the execution of a test are discarded at the end of said test. Additionally, this allows the parent process to survive the abnormal termination (a UOE) or the lack of termination of any given test.

The OpenDCDiag framework monitors the child processes, catches the abnormal termination signals, and reports them. Similarly, the monitor process forcibly terminates the test if it has reached a pre-calculated time limit. Additionally, the monitor reports which threads in the child processes have not reached the end of the execution or the first one to have crashed.

C. Topology enumeration and workload distribution

Enumerating the topology is required to implement both the topology-diverse testing described above and to exhaustively test every sub-unit of the computing device under testing.

As the current implementation of OpenDCDiag is focused on CPUs, the compute sub-units in question are CPU cores. For those, the framework is also required to take into consideration the fact that communication between cores is not uniform, with latencies and collisions increasing, and bandwidth decreasing the further the communication is. Therefore, the framework preferentially distributes the workload such that it does not cross the boundaries of Non-Uniform Memory Access (NUMA) domains, unless testing such crossing is the intent of the test. As an additional benefit, this has allowed OpenDCDiag to scale to thousands of CPU cores. Discovered data is very frequently used for correlating failures to process generation, cache sizes, physical locations on the die, etc.

D. Triaging

An additional benefit of isolating physical sockets while running tests is that, in the majority of cases, identifying the faulty unit is trivial. Even in the case where a fault is not specific to a core but instead is on a block shared by multiple cores on a given die, at least the die or the socket is identifiable.

This in turn makes replacing the unit and returning it to the manufacturer for warranty or further analysis possible.

Further, if the framework is able to identify a specific core or group thereof as the ones containing the fault, system administrators may choose to remove them from orchestration or disable them completely in the operating system, and continue operating the affected node until such a time as full maintenance becomes possible. This is an important benefit for cloud providers and other enterprise customers who may have other, long-running workloads on the same system and who would prefer not to interrupt them, even to perform a migration to another server node.

E. Logging

The need for a machine-readable log output goes without saying. The OpenDCDiag supports three logging formats, the richer of which and current default is based on YAML. The logs provide a final verdict on the test run, either a ”pass” meaning no test could identify a defect, a ”fail” indicating at least one did identify a defective condition, or other words indicating the reason the test run was inconclusive.

To provide a standard output for all tests, the framework offers a few simple functions to test writers, removing the need for each to deploy their own content and agree upon the schema the report is written in. The logging framework must also obey the requirements for survivability outlined above. That in turn implied that it needed to be free of mutex and critical condition locks, as those could lead a problem in one thread that had developed a deadlock condition to propagate to multiple, other threads, thereby making the identification of the root cause less obvious.

The framework is also careful to include in the log output the name of the test that is about to start and asks the operating system to flush the logs to permanent storage before initiating that test. This allows operators to identify which test may be creating such a catastrophic condition that the entire system stops responding or reboots.

F. Random number generation and reproducibility

As described above, one technique for finding unknown defects is to use random data as input. However, using truly random data goes against the ability to reproduce the test conditions at will, a requirement to characterize a detected condition as a hardware defect as opposed to having an external cause. Reproducing the software state also allows hardware designers to analyze the hardware itself to create solutions or mitigations to the problems.

To support both goals, the OpenDCDiag framework provides a Deterministic Pseudo-Random Number Generator (DPRNG). It overrides the C Standard Library functions that operate DPRNGs, thereby ensuring that tests don’t accidentally obtain data not under the framework’s control.

The main DPRNG is seeded using truly random data obtained from the operating system at the start of the execution and is used to seed subsidiary DPRNGs. This allows each software thread to have a deterministic state upon its start. This state is independent of all other threads and of the tests executed in the past.

The state of the main DPRNG is included in the tool’s log output at specific intervals as well as when the test detects an incorrect calculation. This state can be passed back to the tool in a command-line option, replacing the random data used as input in default executions. This causes all the pseudo-random numbers generated to be the same as they were in the previous execution, at the time the DPRNG state was printed.

G. Memory Management

The OpenDCDiag framework also overrides the C Standard Library memory management functions. The overrides add two extra features: first, they ensure all memory they allocate is always initialized with zeroes. This is to ensure accidental use of uninitialized heap memory does not cause tests to have different and often non-reproducible behaviors.

Second, those functions never fail to return memory. This is implemented by causing the child process to abort if it would otherwise have failed a memory allocation, containing the problem.

The framework will always report a failure to allocate memory as an invalid condition in the test log. If the reason is actually the lack of memory, the system administrator may resolve the problem. Similarly, software developers working on the test can be warned of the condition and fix the test.

While memory corruption may be caused by software bugs (such as buffer overruns), given the nature of the workload, it is possible the corruption occurred due to a previous SDC. Therefore, intercepting those conditions allows another class of SDCs to become UOEs.

III. CPU Designer Perspectives (AMD)

Sankar Gurumurthy, Sudhanva Gurumurthi

A. Online Testing in the Era of Small Delay Faults

In recent years, there have been multiple reports from hyperscalars about the occurrence of silent data corruptions (SDCs) in their fleet [1], [2], [24]. These reports bear certain common attributes: the source of the SDCs have been root caused to hardware and the bear the hallmark of intermittent faults. That is, the faults are not transient or random, but repeatable and reproducible. The observed errors have been tied to a few CPUs/cores in a system. While these incidents were originally reported on CPUs, the industry has reported such incidents in other SoCs as well [25]. Another attribute that has been emerging from industry reports is that the root cause of these incidents is a class of defects whose fault characteristic manifests as a small increase in delay of affected sites [26]. We denote these as small delay faults in subsequent sections. An underlying root cause of small delay faults is a small delay defect, which is expected to be more common in deep sub-micron technologies [5]. Moreover, it has been noted that the defects behind these events in hyperscalar infrastructure are no different than the defects normally caught in the manufacturing test [27]. In the subsequent sections, we will provide some reasons as to why some of the defective parts can escape the

manufacturing test flow, how online testing can help address this issue, and our vision for a holistic design+test solution.

B. Small Delay Faults and Manufacturing Testing

1) Small Delay Faults: A small delay fault is the effect of a point defect that manifests as a small increase in delay at the affected site. The term "small" refers to the fact that such defects increase the delay by some fraction of the clock cycle time. Such small delays are benign if the paths through the affected site have enough timing slack. However, if the defect lies on a path that does not have enough slack (e.g., a critical path), these defects can delay the signal transitions sufficiently to cause wrong values to be captured in the path endpoints. Furthermore, a defect that induces a small enough delay to be benign at manufacturing time could degrade into one whose delay magnitude is larger, hence increasing the risk of a failure at a later stage in the lifetime of a part.

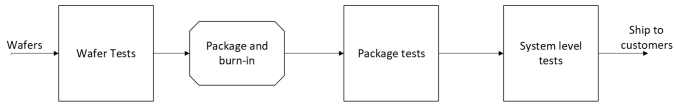

Fig. 2. A simplified example of a manufacturing test flow.

2) Manufacturing Testing Flow: Fig. 2 shows a simple example of how a manufacturing test flow might be implemented. A chip undergoes testing multiple times as part of a manufacturing test flow. For the wafer and package testing phase, specialized test equipment is used that generally includes a stable voltage supply and strong temperature control, in addition to allowing for access mechanisms necessary for structural tests. During system level testing, the part is normally tested in mission mode. The tests that are applied during the manufacturing flow can be broadly classified into two categories: structural tests and functional tests.

3) Detecting Small Delay Faults through Structural Testing: For structural tests, design for test (DFT) techniques are used to partition the design under test. These partitions are then targeted by algorithmic methods to generate tests against specified fault models. Fault models that have been traditionally used include stuck-at and transition delay fault models. Tests generated with stuck-at fault models (as the name implies) are generally efficient in finding defects that cause nodes within a part to get stuck-at a logic '1' or '0' and not defects that only cause a change in delay. Tests generated for transition delay faults are effective at screening defects that cause extra delays. However, transition delay fault tests do not prioritize activating such faults via paths that have a low amount of timing slack, hence making them less effective in detecting small delay faults.

Fault models such as path delay fault models are better suited to be used for test generation that prioritizes detecting small delay faults. However, a path delay fault model requires enumerating a large number of paths which can be prohibitively high for a complex chip, hence making test generation challenging. One potential workaround to this challenge is to limit the number of paths targeted to a more tractable number by limiting it to paths with the least timing slack. However, modern CPUs operate along many points in the voltage/frequency curve and paths of interest can be different at each point point along the voltage/frequency curve, again causing a large excursion in the the number of paths that need to be targeted.

Moreover, structural tests are applied through Automated Test Equipment (ATE) that use highly stable voltage supplies and employ strong temperature controls. Events that can occur in mission mode operations (i.e., when the part is in use in a data center), such as voltage droops and local heating, are less likely to occur during the application of structural tests. On the other hand, the hallmark of a small delay fault is high parametric sensitivity and activation under specific combinations of factors such as voltage, temperature, and workload patterns [26]. These issues can make small delay fault detection challenging through structural testing.

4) Detecting Small Delay Faults through Functional Testing: In CPUs, functional tests are composed of instruction sequences that adhere to the instruction set architecture of the CPU. Functional tests are commonly applied in the manufacturing test flow at part of the system level testing phase (Fig. 2). Functional tests can also be applied during the wafer or package test phase. However, there might be more constraints on functional tests during this phase. For example, they might need to be able to fit in the internal caches of the CPU [28], [29].

Functional tests are run in mission mode and hence are capable of recreating the voltage droop and thermal events to elicit a fault. However, they are a limited representation of all the applications that can possibly run on a modern CPU. To be effective, functional tests might still need to be handcrafted by experts to recreate the interesting scenarios, making them highly labor intensive and risks not covering all fault models of interest due to human blind spots. Automation tools for coverage evaluation and other necessary activities for test generation also lag behind those available for structural tests due to the challenging nature of this problem.

C. Detecting Small Delay Faults through Online Testing

In addition to the shortcomings of the manufacturing tests in catching small delay faults, some defective parts can escape testing because of insufficient burn-in/aging of latent defects. One way to address these test escapes is to augment manufacturing testing with online testing to weed out defective parts from server fleets to reduce the risk of data corruption in the field.

There are two general approaches to online testing of fleet parts: (a) in-production testing, (b) out-of-production testing

In-production testing is carried out on hardware running in the fleet, where the tests are run alongside normal workloads. The amount of disruption caused to mission workloads needs to be minimized to reduce the overhead to normal fleet operation. Even though the tests might be isolated from the software perspective from the rest of the system, the tests are still coupled with other workloads on the system from

the hardware perspective since they are sharing resources and can hence impact performance (e.g., increase tail latency of online services). Hence, such tests need to be carefully crafted to minimize the use of shared resources, such as memory bandwidth. In addition, tests should also spend limited time on the CPU cores under test. For example, tests might need to be limited to running at the most hundreds of milliseconds at a time [30].

In out-of-production testing, the tests are carried out on parts that have been removed from fleet operations for various reasons, such as for firmware or kernel updates. Some portion of this downtime can be allotted to testing. In these cases, the tests can utilize the entire system without concern for competing applications sharing the hardware resources. Typically, the opportunity to run such pervasive tests tend to be on the order of a few minutes.

There are challenges and advantages in both in and out-of-production testing. Since in-production testing limits the resources available to tests, crafting effective tests that can run under those constraints is a key challenge. Out-of-production tests do not have these constraints, but can only be run when a system is brought out of a production fleet. This might not be a common occurrence, and many weeks or even years can elapse between such testing windows. In-production testing, on the other hand, can be run more frequently. A key requirement to make online testing effective is that such tests must provide high coverage in catching defects. Recent research [13] suggests that a hardware-in-the-loop approach (unlike random hardware agnostic approaches) can facilitate the generation of high-coverage functional tests directed on specific hardware structures, tied to specific fault models, and can be tailored to either online testing scenario.

D. Is Testing Alone Enough to Address Small Delay Faults?

Testing is important but unlikely to be a silver bullet to address small delay faults. Exhaustive testing is challenging due to the size of modern processors and the high parametric sensitivity of the root causes of these faults. Supplementing testing with resilient design (aka Reliability, Availability, Serviceability or "RAS") can provide a more holistic approach. While testing aims to reduce DPPM, RAS can reduce data corruption risks from test escapes. RAS can also improve observability of existing system level tests, hence enhancing manufacturing tests. Targeted RAS techniques that are cognizant of root causes and whose benefits can be estimated a-priori systematically and quantitatively are valuable, as evidenced from previous reliability challenges faced by the industry [31].

E. Summary

Small delay defects pose a key reliability challenge that requires leveraging the best practices of structural and functional testing, addressing their individual gaps, and driving research and innovation in online testing. Combining test and RAS provide a holistic solution to remediating reliability challenges and enabling resilience at scale.

IV. CPU Designer Perspectives: ARM

Amr Haggag

A. Sensor framework for predictive SDC maintenance

Due to high demands imposed by applications such as AI, data center operators are required to maintain consistent operations with little to no interruption of service. Datacenter diagnostic patterns, as discussed in an earlier section, are often deployed to measure the health of an SoC post-manufacturing and in-field to mitigate SDC incidents that can cause such service interruption. This can help reduce SDC incidents from 1000's to 10's DPPM. But there is a need to do more and this is where on-chip sensors can help. Having sensors to detect droops or aging or thermal events on chip all of which can cause SDC helps address the residual DPPM issues. For this reason, scalable, customizable sensor frameworks that can monitor and adapt throughout the useful life of the device are needed. Datacenter providers can collect this sensor data and correlate it to SDC event incidents. This would be the training phase and would employ ML methods. This is followed by a deployment phase where a combination of sensor readings are deemed too risky for SDC and can flag an SoC for additional checking through diagnostic patterns before it fails. One can also use these sensors to adjust operating conditions to give more voltage margin, for example, if a part has aged beyond normal expectation. This sensor framework can work hand in hand with data center diagnostic patterns to enable predictive maintenance. In summary, sensors provide the capability and scope to monitor and adapt throughout the useful life of the device, enabling both design optimization and predictive maintenance.

B. DFX Methods to address SDC in Advanced Nodes

DFX is a critical item in delivering production-quality silicon. But there are also key components of DFX that can help improve SDC failure rates. A strong DFX methodology can help remove much of the burden from post-silicon and in-field mitigations. Starting with DFT, high coverage must be obtained for not just stuck-at, but also transition and path delay coverage. As discussed in the prior section small delay defects are a large contributor to SDC. Tests must also be cell- and layout-aware. In addition, one must enable functional ATE tests that are architecture-aware, given structural tests don't fully cover all the design sensitivities. DFD is also critical to enabling a good handle on SDC. When SDC fails are triaged back to the vendor, good DFD techniques such as embedded logic analyzers (ELA) for logic, extra margin adjust (EMA) for memories, scan and array dumps, on-die eye monitors (ODEM) for high-speed I/Os and even sensors as discussed previously can facilitate fast debug and a correlation of fails to marginalities on-die. Many fails are also latent, therefore being able to screen for latent defects is key, and this is where DFR/DFM comes in. While DFR ensures proper margins are in place for aging and EMIR, sufficient protection in place for SER FIT, etc., we must also enable HVS (high voltage stress) screen capability where a part can be screened at 1.5x Vmax without latch-up. This has

become challenging on advanced nodes and special foundry deck rules must be employed. If all these DFX items are done properly, it will help guarantee part resilience to SDC and reduce the occurrence likelihood. This in turn releases much of the burden from post-silicon screens.

V. Conclusion

Tackling the issues caused by silent data corruptions (SDCs) is essential for preserving the reliability and efficiency of large-scale computing infrastructures. Contributions from the three leading CPU Designers (Intel, AMD, Arm) towards the emerging challenge of Silent Data Corruptions (SDCs) are summarized in this paper. By implementing various strategies and technologies, these industry leaders show a dedicated effort to reduce the effects of SDCs, thereby ensuring dependable hardware performance in cloud data centers, supercomputers, and edge systems. Intel's contribution focuses on the OpenDCDiag framework for defects detection. AMD's contribution emphasized the importance and difficulty of detecting small delay defects. Arm's contribution main point is the combination of DFX and on-chip sensors in predictive maintenance during the lifetime of chips. The industry concurs that it is critical to identify the primary sources of errors that can silently affect program execution and to develop innovative methods for modeling and detecting SDCs; recent efforts to raise awareness and engage the research community reflect this major need [19], [20].

Acknowledgment

The authors affiliated with the University of Athens have been supported by Meta Platforms, AMD Research, the Open Computer Project (OCP), as well as the European Union's Horizon Europe research and innovation programme under grant agreement No 101093062 (Vitamin-V). 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] 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*, 2021, pp. 9–16.

[2] 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

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

[4] 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.

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

[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] 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.

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

[9] ——, "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.

[10] 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

[11] 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.

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

[13] N. Karystinos, O. Chatzopoulos, G. Fragkoulis, G. Papadimitriou, D. Gizopoulos, and S. Gurumurthi, "Harpocrates: Breaking the silence of cpu faults through hardware-in-the-loop program generation," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2024.

[14] 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

[15] 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.

[16] 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.

[17] 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.

[18] D. Agiakatsikas, G. Papadimitriou, V. Karakostas, D. Gizopoulos, M. Psarakis, C. Belanger-Champagne, and E. Blackmore, "Impact of voltage scaling on soft errors susceptibility of multicore server cpus," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 957–971. [Online]. Available: https://doi.org/10.1145/3613424.3614304

[19] "Ocp's server resilience initiative: Sdc academic research awards announced!" https://www.opencompute.org/blog/ocps-server-resilience-initiative-sdc-academic-research-awards-announced, accessed: 2024-06-21.

[20] "Announcing the winners of the 2022 silent data corruptions at scale request for proposals," https://research.facebook.com/blog/2022/6/announcing-the-winners-of-the-2022-silent-data-corruptions-at-scale-request-for-proposals/, accessed: 2024-06-21.

[21] "Data Center Silent Data Errors," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/788204/data-center-silent-data-errors-technical-paper.html

[22] "AMD, OpenFieldHealthCheck," https://github.com/amd/Open-Field-Health-Check.

[23] "OpenDCDiag." [Online]. Available: https://github.com/opendcdiag/opendcdiag

[24] 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*, 2023, pp. 216–230.

[25] Y. He, M. Hutton, S. Chan, R. de Gruijl, R. Govindaraju, N. Patil, and Y. Li, "Understanding and mitigating hardware failures in deep learning training accelerator systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2023.

[26] P. G. Ryan, I. Aziz, W. B. Howell, T. K. Janczak, and D. J. Lu, "Process defect trends and strategic test gaps," in *2014 International Test Conference*. Seattle, WA, USA: IEEE, Oct. 2014, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/7035276/

[27] M. Shamsa and D. Lerner, "Defect mechanisms responsible for silent data errors," in *2024 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2024, pp. 1–5.

[28] S. Gurumurthy, R. Vemu, J. Abraham, and D. Saab, "Automatic generation of instructions to robustly test delay defects in processors," in *12th IEEE European Test Symposium (ETS'07)*, 2007, pp. 173–178.

[29] S. Gurumurthy, M. Pratapgarhwala, C. Gilgan, and J. Rearick, "Comparing the effectiveness of cache-resident tests against cycleaccurate deterministic functional patterns," in *2014 International Test Conference*. IEEE, 2014, pp. 1–8.

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

[31] S. Mukherjee, *Architecture design for soft errors*. Morgan Kaufmann, 2011.