Evaluating ECC for Cache Reliability Under Multi-Bit Upsets: A Design Space Exploration

Foteini Kotsimpou George Papadimitriou Dimitris Gizopoulos
University of Athens, Greece
{fotkotsimpou | georgepap | dgizop}@di.uoa.gr

***Abstract*—Cache memory reliability is a critical concern in modern processor architectures, particularly in systems deployed in fault-prone environments or at large scale. As modern computing systems increasingly rely on cache memories for performance, ensuring their reliability against transient faults (soft errors) is crucial. Several protection schemes, such as parity or error-correcting codes (ECC), have been proposed to enhance cache robustness. However, these techniques introduce trade-offs in terms of performance and effectiveness, which are particularly relevant in RISC-V-based CPUs due to their flexibility and modular design. This paper presents a comprehensive characterization and design space exploration of error protection mechanisms (parity and Single Error Correction Double Error Detection - SECDED - ECC) in the cache memories of RISC-V CPUs. We employ statistical fault injection at the microarchitecture level to evaluate the impact of transient faults on system reliability, analyzing their effects across system layers in a full-system setup using the gem5 simulator, the only environment that supports the execution of long workloads. By comparing caches with and without protection, we provide insights into error resilience, overheads, and the effectiveness of protection schemes. Our study can drive the selection of appropriate protection strategies for various RISC-V implementations, ranging from embedded systems to high-performance computing platforms. Our findings offer valuable guidance for the design of robust and efficient RISC-V-based systems operating under reliability constraints.**

***Index Terms*—Reliability, RISC-V, ECC, microprocessors, caches, microarchitecture, fault injection, silent data corruption.**

I. Introduction

Cache reliability is a critical concern in modern microprocessors, particularly as technology scaling increases susceptibility to transient faults and soft errors [1], [2]. In RISC-V architectures, which emphasize extensibility and open-source innovation, designers have flexibility in implementing various cache protection schemes. Moreover, new hardware-based protection solutions continue to emerge, aiming to improve essential aspects such as performance, energy efficiency, and area overhead, thereby enhancing the overall effectiveness of protection schemes [3]–[5]. However, selecting the optimal protection mechanism requires careful consideration of trade-offs between reliability, performance, and hardware overhead.

A widely adopted approach to mitigating soft errors in cache memories is the uniform integration of error detection codes (EDC) and error correction codes (ECC) across all cache blocks [3]. In these methods, each cache block is augmented with an EDC and/or ECC, ensuring that every read or write operation includes error detection/correction or the encoding of EDC/ECC, respectively. Typically, parity codes serve as the simplest form of error detection, while Hamming [6] and Hsiao [7] codes are the most common ECC techniques, enabling single-bit error correction and double-bit error detection (SECDED). These methods are usually used in cache memories of modern microprocessors due to the substantially low overhead associated with the storage and complexity of ECC circuitry [8]. Specifically, some microprocessors employ lightweight error detection codes (EDC), such as parity, in timing-critical L1 caches, while delegating error correction to lower levels of the cache hierarchy. This approach is a common protection strategy in modern microprocessors to reduce the latency overhead associated with ECC codes, during read operations [9], [10].

As microprocessor designs move towards more open and customizable architectures, ensuring cache reliability in RISC-V CPUs is becoming an increasingly important challenge. Unlike traditional proprietary ISAs, RISC-V enables tailored microarchitectural decisions, including cache structures and protection mechanisms. However, this flexibility also means that there is no one-size-fits-all solution for cache reliability. Soft errors induced by radiation, aging effects, and manufacturing variations pose significant reliability risks, especially in safety-critical systems, edge computing, and high-performance computing environments [11]. Given the increasing adoption of RISC-V in automotive, IoT, and HPC applications, a systematic study of cache protection mechanisms tailored for RISC-V architectures is necessary.

This paper aims to fill this gap by exploring the most prevalent protection techniques, assessing their effectiveness, and providing practical insights for designers. We employ a design space exploration approach, incorporating statistical fault injection to assess the impact of the ECC technique SECDED on single-bit, double-bit, and triple-bit faults and present a comprehensive investigation of fault behavior, comparing a non-protected with a SECDED scheme. All comparisons are conducted using single-, double-, and triple-bit fault injections. To the best of our knowledge, this is the first study to evaluate the impact of both single- and multi-bit flips on the dominant protection schemes for cache memories. By quantifying the reliability gains and overheads of different protection strategies, this work offers practical insight for RISC-V architects in selecting or assessing cache protection strategies.

The main contributions of this paper are enumerated below:

1) We first quantify the rate at which double- and triple-bit upsets lead to failures and assess the fraction of faults that are completely handled at the hardware level (i.e., they are getting masked due to microarchitectural operations or corrected through SECDED) and do not propagate further.

2) We identify the proportion of faults that propagate beyond the hardware-level detection but do not impact software execution (masked at the software level).
3) We finally showcase whether specific software workloads or access patterns make errors more or less visible and determine if certain workloads are more prone to failure due to specific memory access patterns.

II. Background

A. Reliability Concepts

The Architectural Vulnerability Factor (AVF) quantifies the probability that a fault within a microprocessor's hardware structure will result in a visible program error. AVF considers the interplay between the microarchitecture, program structure, and input data, enabling a comprehensive vulnerability assessment across multiple layers, including microarchitecture, architecture, and software [12]. Since AVF is technology-independent, it provides a holistic measure of system vulnerability by encompassing all stages of fault activation and propagation from hardware to software [13]. For example, in an out-of-order microprocessor, a fault in the physical register file may have no impact on execution if discarded due to a pipeline flush. Similarly, even if a fault propagates to the software layer, the affected data might be overwritten or ignored by the algorithm, ensuring correct program execution. Thus, faults can be masked at the hardware or software level. Computer architects typically rely on AVF to determine when and where to integrate redundancy in system designs, especially when evaluating runtime reliability techniques and conducting cross-layer analyses of fault-tolerant mechanisms (e.g., [14], [15]). To this end, in this study, we report AVF values for all results to provide insights into system reliability and fault resilience.

B. Statistical Fault Injection (SFI)

Designers typically rely on Statistical Fault Injection (SFI) [16] or analytical approaches such as Architecturally Correct Execution (ACE) analysis [12] to evaluate a program's resilience to transient faults (i.e., soft errors). ACE analysis specifically applies to transient faults, whereas fault injection can assess both transient and permanent faults, by simulating program execution under fault conditions. Both methods aim to estimate the AVF for hardware structures, yet they differ in accuracy, efficiency, and complexity. SFI provides high accuracy by directly observing program behavior, but it requires multiple executions to achieve statistical confidence, making it computationally expensive. In contrast, ACE analysis is faster but requires significant development effort and tends to overestimate AVF [17]. As a widely used reliability evaluation method, SFI enables full-system AVF estimation by analyzing program outputs affected by injected faults. It offers flexibility in accuracy, depending on sample size, and generates failure data through simulation. However, its primary drawback is the high computational cost, from extensive simulations, which may be impractical depending on the model's complexity.

III. Experimental Setup

Through an extensive SFI and simulation-based study, we gather valuable insights that highlight the accuracy improvements our approach brings to vulnerability estimation in modern CPUs. Our analysis is based on 9 long-running benchmark programs, each simulated in an out-of-order (OoO) RISC-V microprocessor, evaluating both L1 instruction and data caches. We conduct experiments on gem5 simulator by injecting one, two, and three faults per run for each cache component, with low error margin and high confidence level, while assessing different protection mechanisms, including parity and SECDED ECC. Our implementation extends cache protection in gem5 by supporting both single-bit parity and SECDED at configurable granularity levels, block- and word-level, making the system flexible for various reliability-performance trade-offs. For each protection scheme, three key functions are implemented: one to encode and store parity or SECDED check bits during data initialization or write operations, another to update these parity/check bits on every write, and a third to verify, with correction in the case of SECDED, on read.

A. Experimental Platform

Among the various levels of abstraction available for system modeling, the microarchitecture level is the only one that provides sufficient hardware detail while also enabling full-system simulation, including the operating system [18]–[22]. Although Register Transfer Level (RTL) modeling offers a more hardware-accurate representation, its low simulation throughput makes it impractical for executing large-scale fault injection studies that involve a complete system stack, including the operating system and I/O interactions. Our microarchitectural modeling is built on the gem5 simulator, a state-of-the-art, flexible, full-system cycle-level simulator [23]–[25]. The gem5 framework provides comprehensive RISC-V ISA support and includes a detailed out-of-order core implementation. Table I outlines the configuration of the experimental CPU core used in our gem5 simulations. For fault injection and reliability assessment, we employed the GeFIN fault injection framework [26], which we extended by integrating a new fault generator capable of injecting spatial multi-bit upsets with varying geometries and cardinalities during system execution. For each structure, 500 randomly generated faults are injected, based on the statistical fault sampling method that follows the

TABLE I
Major Simulator Configurations.

Parameter	Value
ISA	RISC-V
Pipeline	64-bit OoO (8-issue)
L1 Instruction Cache	32KB, 64B line, 128 sets, 4-way
L1 Data Cache	32KB, 64B line, 128 sets, 4-way
L2 Cache	1MB, 64B line, 2048 sets, 8-way
Physical Register File	128 Int; 128 FP
LQ/SQ/IQ/ROB entries	32/32/64/128
IQ entries	64
Branch Predictor	Tournament Predictor
Branch Target Buffer	Direct-Mapped, 4K entries
RAS Size	16 entries

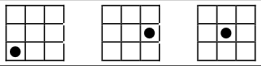
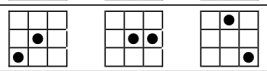
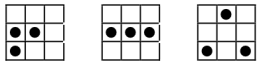
Category	Error Bit Pattern Example
Single-bit Fault	
Double-bit Fault	
Triple-bit Fault	

Fig. 1. Examples of multi-bit upset categories in a 3x3 cluster used in our experiments.

widely used formulation from [16], ensuring an error margin below 5% with a 99% confidence level. Fault injection at the microarchitecture level provides high observability, enabling precise tracking of where faults occur (e.g., distinguishing between kernel mode vs. user mode or whether corrupted data was used or discarded). Additionally, this approach allows for a detailed analysis of system-wide effects, which we further explore in the following subsection.

B. Fault Modeling (Single- and Multi-bit Flips)

For the single-bit fault, we follow the widely used formulation from [16] injecting a single bit-flip for each execution based on the uniform distribution. On the other hand, we augment this formulation for spatial multi-bit fault as described below: spatial multi-bit faults can impact neighboring memory cells that share a p-well or n-well, as well as other cells within close proximity. Based on prior research [1], [27], [28], we adopt the fault cluster model introduced in [27], similar to the approach in [1]. In this model, for a cluster size of X rows by Y columns, the generator randomly assigns N fault locations (bit flips) within the defined region. The fault cluster is then randomly placed inside the SRAM array, marking the locations where faults will be injected. This framework allows the configuration of cluster sizes and fault distributions to reflect realistic patterns observed in modern fabrication technologies. Each cluster's vulnerability is assessed individually, and its contribution to the overall AVF is weighted based on technology-specific characteristics. This level of customization enables evaluations for both current and future technology nodes. Our experiments introduce single-, double-, and triple-bit faults within a 3×3 cluster (fault rates for larger clusters, such as quadruple-bit faults, are negligible, as reported in [1], [27]).

To conduct fault injection, we generate a fault mask containing multi-bit errors that are applied to the CPU hardware. In each experiment, we isolate a 3×3 region, randomly select faulty cells within this cluster, and inject faults accordingly. These adjacent bit flips closely resemble fault patterns observed in accelerated beam testing [29]–[32], where multi-bit errors typically manifest in neighboring memory blocks [1]. Fig. 1 presents examples of single-bit, double-bit, and triple-bit faults within a 3×3 cluster, generated by our fault injection model. Unlike the MBU (Multiple Bit Upset) coding approach from [27], which defines the cluster as the smallest possible region encompassing all faults, our methodology allows faults to fit within smaller clusters (e.g., some double-bit errors could be accommodated within a 2×2 cluster). This approach provides a more realistic fault distribution, ensuring that sub-clusters of varying sizes are incorporated into our analysis.

IV. Methodology & Experimental Results

A. Total AVF for Every Number of Bit-Flips

Fig. 2 shows the total AVF for single-bit, double-bit, and triple-bit fault injections. This graph clearly shows that double-bit and triple-bit provide higher AVF than single-bit flips in both L1 data and instruction caches. Surprisingly, L1 instruction cache shows that the impact of double-bit and triple-bit flips are significantly larger than in L1 data cache, suggesting that L1 instruction cache is clearly more prone to multi-bit upsets and may require stronger protection than L1 data cache.

B. SECDED vs. No-Protection AVF

Fig. 3 presents the AVF for double-bit flips in both the L1 data and instruction caches, while Fig. 4 shows the AVF for triple-bit flips. These figures clearly illustrate that SECDED provides effective protection (i.e., correction) against both double- and triple-bit flips; however, there is still room for improvement. The data in these graphs reveal that the SECDED AVF (green bars) is significantly lower than the No-Protection AVF (red bars), demonstrating that SECDED can substantially reduce failure rates. Specifically, in Fig. 3, SECDED reduces the AVF in the L1 data cache more effectively than in the L1 instruction cache, where the SECDED AVF remains relatively uniform across all benchmarks used in this study. Interestingly, in Fig. 4, the L1 instruction cache exhibits a lower SECDED AVF for triple-bit fault injections. This suggests that SECDED becomes more efficient at correcting bit flips as the number of adjacent faults increases beyond double-bit flips. One possible explanation is that multiple-bit flips, particularly triple-bit flips, are more likely to be distributed across different cache blocks or words within a cache block rather than being clustered together as in double-bit flips. Overall, across all benchmarks, SECDED reduces the failure rate by a factor of 2–6x, confirming its effectiveness in mitigating the impact of multiple-bit faults.

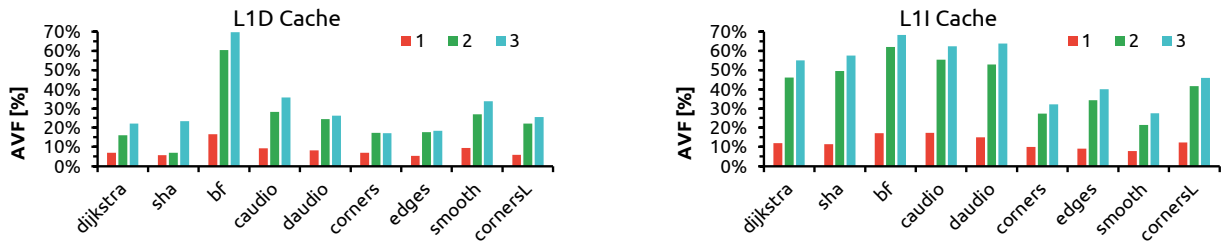


Fig. 2. AVF for all fault injection (single-bit, double-bit, and triple-bit).

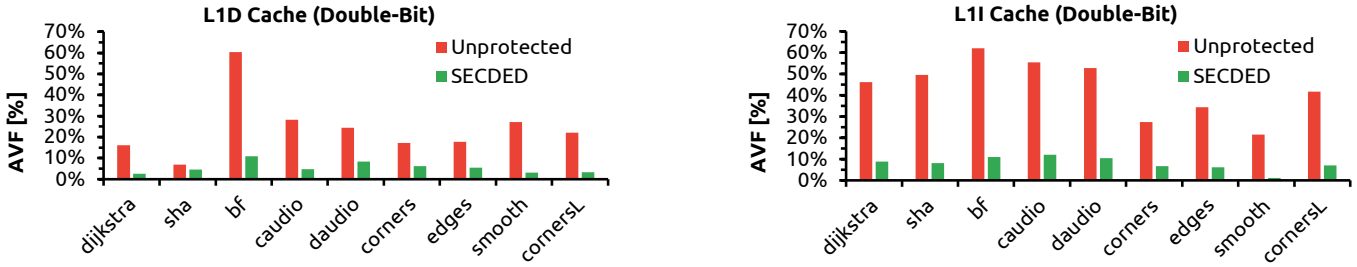


Fig. 3. AVF for double-bit flips for both L1 data and instruction caches.

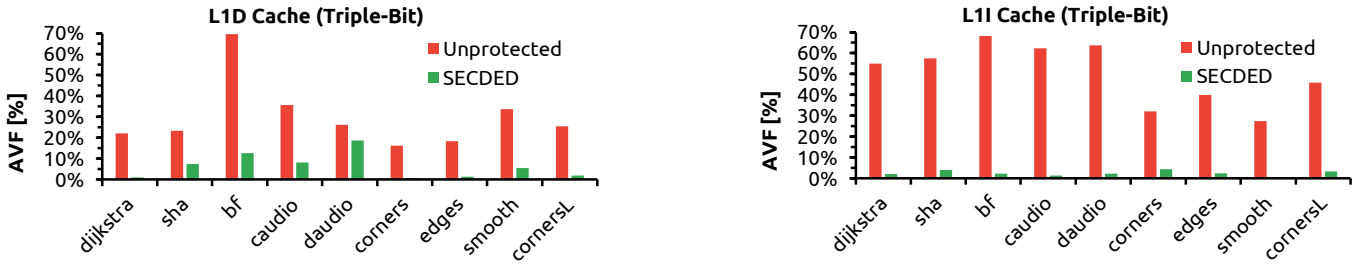


Fig. 4. AVF for triple-bit flips for both L1 data and instruction caches.

C. Overall Fault Distribution

Fig. 5 shows the breakdown of the overall fault distribution using SECDED for double- and triple-bit fault injections and for both L1 data and instruction cache memories. Specifically, for each pie chart, we can see five (mutually exclusive) categories:

1) **HW Masked Faults** category includes any fault in the L1 data or instruction cache that occurs in an invalid cache block.
2) **Non-Critical Corrections** category includes all faults that are corrected by SECDED, but the affected words (whether instructions or data) are ultimately masked at the software level. This means that even if this subset of faults were not corrected by the protection scheme, they would never impact the program's execution.
3) **Detectable but Uncorrectable Errors (DUE) masked in SW** category includes faults that were detected but could not be corrected; however, they were masked by the software. Thus, although this category consists of DUEs, the program's execution remains correct.
4) **Detectable but Uncorrectable Errors (DUE) causing failure** category includes faults that were detected but could not be corrected, ultimately affecting the program's execution.
5) **Needed Corrections** category includes all faults that were ultimately corrected by SECDED. This correction is essential for the correct execution of the program. If faults in this category were not corrected by the protection scheme, they would impact the program's execution.

These categories offer key insights into how faults impact the system. Each chart highlights how injected faults are handled, including whether they are masked, corrected, or result in failures. In the L1 data cache with double-bit faults, 31.9% of the faults are masked by the hardware, i.e., they do not propagate further and no SECDED operation was needed. 35.1% of the faults are categorized as non-critical corrections, where SECDED applied error correction even though these faults would have been masked later. 8.5% of faults were detectable but uncorrectable and got masked in software, while only 5.1% resulted in failure. The remaining 19.1% are real needed corrections, meaning SECDED successfully prevented potential failures. This breakdown highlights the combined effectiveness of HW/SW masking and SECDED, although a significant portion of corrections could have been avoided.

For the L1 instruction cache with double-bit faults, only 6.8% are hardware-masked, showing higher fault propagation.

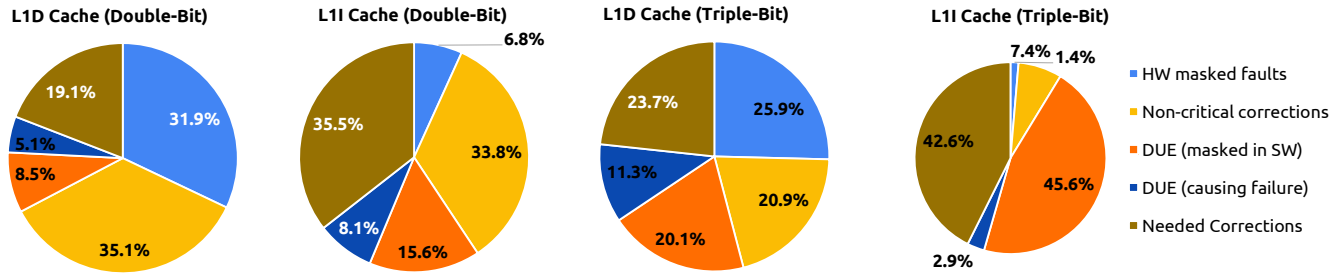


Fig. 5. Breakdown of overall fault distribution using SECDED for double- and triple-bit fault injections and for both L1 data and instruction caches.

Non-critical corrections account for 33.8%, while 15.6% of the faults passed the HW/SW interface but were masked at the software level. 8.1% of faults resulted in failures, a slight increase compared to L1 data cache. Meanwhile, a significant 35.5% are genuine SECDED corrections. This suggests that the instruction cache is more vulnerable to double-bit faults that do not get masked across the layers, resulting in more DUEs. However, SECDED corrects a significant amount of them, bringing the rate of failures down to the same level as the L1 data cache.

For triple-bit faults in L1 data cache, hardware masking increases to 25.9%., than in the double-bit scenario. A small 20.9% portion of faults fall under non-critical corrections, while a similar 20.1% portion of faults fall under software-masked DUEs, suggesting that SECDED still attempts to correct many errors that may not affect execution at the same rate as the double-bit scenario. However, failures rise to 11.3%, more than twice the double-bit rate. The proportion of needed (i.e., critical) corrections is 23.7%, indicating that SECDED still handles many triple-bit faults effectively.

The L1 instruction cache exhibits a notably different pattern, with just 1.4% hardware masking for triple-bit faults. This percentage is much lower than the L1 data cache in the same scenario; however, it aligns with our observations for the double-bit case. Non-critical corrections decrease to 7.4%; however, we can see a notable increase in the software-masked DUEs from 15.6% to 45.6%. Only 2.9% of the faults led to failure, while 42.6% were needed corrections. This suggests that the L1 instruction cache is more effective in correcting triple-bit than double-bit faults, which is attributed to more than 95% of all faults. Nonetheless, nearly half of these faults did not need to be corrected, as they would get masked at the software level.

Overall, these results demonstrate that L1 instruction cache is more vulnerable to both double- and triple-bit faults than L1 data cache. The effectiveness of hardware masking decreases with triple-bit faults, allowing more faults to propagate. While SECDED appears to successfully handle many double-bit and triple-bit upsets in L1 instruction cache, a significant portion would have been masked at the SW level. While in the L1 data cache the corrections are less compared to the L1 instruction cache, resulting in more failures, the corrections that get masked are also decreased. Notably, non-critical and software-masked detectable but uncorrectable errors dominate the L1 instruction cache, exceeding 50% for both fault types. This emphasizes the need for enhanced error protection mechanisms that improve the reliability while avoiding redundant corrections.

To further assess SECDED's capabilities, and provide valuable insights for future designs, we analyze two additional metrics: the effectiveness and the efficiency of the most prevailing protection scheme in cache memories.

Fig. 6 shows SECDED's effectiveness against double-bit and triple-bit faults across all the studied benchmarks. The effectiveness is measured by the reduction in AVF compared to an unprotected baseline. This way, we can measure the magnitude, in which SECDED can correct the failures from the double- and triple-bit faults. At the double-bit faults, SECDED appears to be slightly more effective in L1 instruction cache against the L1 data cache across most benchmarks. This insight aligns with the observations we made in Fig. 5, in which although the instruction cache had more failures compared to the data cache, the corrections balanced the results. At the triple-bit faults, the difference becomes more pronounced, with the SECDED in the L1 instruction cache being more effective than the data cache in all benchmarks but one. In this figure, it is also clear that the SECDED in L1 data cache shows less consistent effectiveness, with one benchmark performing notably worse in both fault scenarios.

Efficiency, shown in Fig. 7, is calculated based on the ratio of needed to non-critical corrections—higher efficiency corresponds to fewer unnecessary corrections. In the double-bit case, SECDED shows similar results in both cache memories with the instruction cache appearing as slightly more efficient in most benchmarks. These results support Fig. 5 where the instruction cache had nearly twice the needed corrections of the data cache, while non-critical corrections increased by less than 1.5%. At the triple-bit faults, efficiency improves further in L1 instruction cache, which is virtually 100% in most cases. These results derive from a sharp drop in non-critical corrections compared to the double-bit scenario for the instruction cache, but not that substantial for the data cache. These results give SECDED a clear edge in the L1 instruction cache, offering valuable insight into its effectiveness and use in both caches.

V. Related Work

Sadler and Sorin in [33] presented a comprehensive evaluation of error correction schemes for L1 data caches, comparing parity-protected write-through L1 caches with ECC-protected L2 caches against ECC-protected L1 caches. They introduced the Punctured ECC Recovery Cache (PERC), which combines the advantages of both schemes to enhance performance and reliability. Yoon and Erez in [3] proposed Memory Mapped ECC, a novel technique that reduces the cost of error protection in last-level caches by storing ECC bits within the memory hierarchy as data. This approach minimizes hardware overhead while maintaining strong error correction capabilities. Weaver et al. in [14] explored techniques to reduce the soft error rate in high-performance microprocessors. They introduced methods to selectively squash instructions experiencing long delays and proposed modifications to error detection logic to differentiate between true and false errors, thereby enhancing processor reliability. Zhang et al. in [4] investigated adaptive ECC techniques aimed at improving the yield and reliability of flash memories. Their work focuses on dynamically adjusting error correction strength based on the error characteristics, thereby optimizing performance and reliability. These studies collectively contribute to the advancement of error correction methodologies in various memory hierarchies, offering insights into balancing performance, area, and power considerations. However, none of them presented a comprehensive and detailed study that compares the most prevalent EDC and ECC schemes

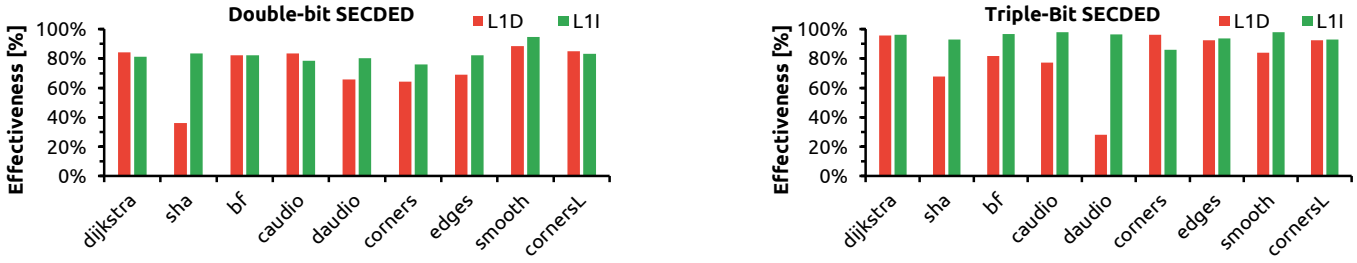


Fig. 6. Protection effectiveness per benchmark for SECDED for double- and triple-bit fault injections and for both L1 data and instruction caches.

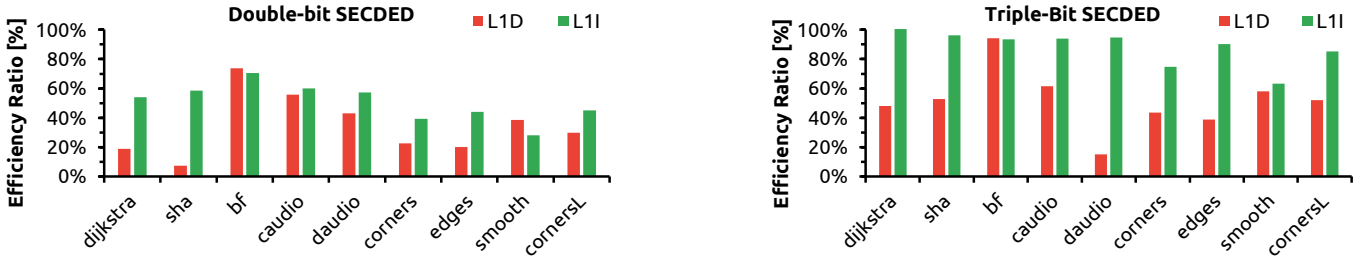


Fig. 7. SECDED efficiency ratio per benchmark for double- and triple-bit fault injections and for both L1 data and instruction caches.

that are commonly used in modern microprocessor chips. Alam et al. in [34] introduced Parity++, a lightweight error correction scheme for last-level caches that provides single error detection for all messages and single error correction for a subset of messages. This approach achieves approximately 9% lower storage overhead and reduced error detection energy compared to traditional SECDED codes. Qureshi and Chishti in [35] proposed FLAIR (FLexible And Introspective Replication), a technique that enables ultra-low voltage operation of SECDED-based caches by performing two-way replication during testing to maintain robustness and disabling lines with multi-bit failures after testing. FLAIR achieves a Vmin of 485 mV, similar to ECC-8, while incurring minimal storage overhead. Ko et al. in [36] conducted a quantitative evaluation of parity-protected L1 data cache design alternatives to maximize protection against soft errors. They developed an algorithm to accurately model data vulnerability in caches with various parity protection configurations and formulated guidelines for designing power-efficient and reliable L1 data caches. These studies further contribute to the advancement of error correction methodologies in memory hierarchies, offering insights into balancing performance, area, and power considerations. Our study is orthogonal to these works, augmenting the landscape by providing insights into the predominant protection schemes. Manoochehri et al. in [37] presented PARMA+, an accurate and efficient model for estimating cache FIT rates under multi-bit transient faults. Their work emphasizes the importance of modeling realistic fault patterns and accounting for spatial failure dependencies.Wilkening et al. in [38] introduced an extension of ACE analysis, to estimate the AVF under spatial multi-bit transient faults. They demonstrated that multi-bit AVF behaves differently from single-bit AVF, emphasizing the need for distinct modeling. George et al. in [39] revisit traditional AVF estimation by proposing a scalable probabilistic fault model that accounts for spatial multi-bit upsets in SRAM arrays. While conventional single bit-flip models can significantly overestimate AVF, by using detailed fault injection and microarchitectural simulation, they provide tighter and more realistic vulnerability bounds.

VI. Conclusion & Future Work

In this paper, we conducted a comprehensive evaluation of cache reliability by analyzing the impact of single-, double- and triple-bit faults (common in modern manufacturing nodes) under a SECDED protection scheme. Through SFI, we quantified the fraction of errors that are masked at the hardware and software levels, providing insights into how faults propagate through the system. Our findings highlight that while SECDED effectively corrects single-bit errors and reduces failures, in multi-bit injections, a large portion of corrections involve non-critical faults already masked at higher layers, highlighting the need for diligently designed stronger protection in more demanding applications. Future work could explore alternative error correction schemes beyond SECDED, expand the analysis to other architectural components, and consider more complex workloads and system configurations. We have also implemented the technique of bit interleaving and plan to evaluate its effectiveness in combination with ECC for further enhancements in fault resilience.

Acknowledgment

Research funded by HFRI through grant No 16973 (RE-DESIGN), supported by research gifts from Meta, AMD, OCP (Open Compute Project), as well as the European Union's Horizon Europe programme under grant No 101093062 (Vitamin-V), the Chips JU grant No 101097224 (REBECCA), and the EuroHPC JU grant No 101202459 (DARE SGA1). 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] 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.

[2] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone, "Open-source ip cores for space: A processor-level perspective on soft errors in the risc-v era," *Computer Science Review*, vol. 39, p. 100349, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013720304494

[3] D. H. Yoon and M. Erez, "Memory mapped ecc: low-cost error protection for last level caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 116–127. [Online]. Available: https://doi.org/10.1145/1555754.1555771

[4] S.-K. Lu, S.-X. Zhong, and M. Hashizume, "Adaptive ecc techniques for yield and reliability enhancement of flash memories," in *2016 IEEE 25th Asian Test Symposium (ATS)*, 2016, pp. 287–292.

[5] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6. [Online]. Available: https://doi.org/10.1145/2593069.2593229

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

[7] M. Y. Hsiao, "A class of optimal minimum odd-weight-column sec-ded codes," *IBM J. Res. Dev.*, vol. 14, no. 4, p. 395–401, Jul. 1970. [Online]. Available: https://doi.org/10.1147/rd.144.0395

[8] S. Mukherjee, J. Emer, T. Fossum, and S. Reinhardt, "Cache scrubbing in microprocessors: myth or necessity?" in *10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004. Proceedings.*, 2004, pp. 37–42.

[9] N. Quach, "High availability and reliability in the itanium processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, 2000.

[10] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 197–209.

[11] A. Bosio, I. O'Connor, M. Traiola, J. Echavarria, J. Teich, M. A. Hanif, M. Shafique, S. Hamdioui, B. Deveautour, P. Girard, A. Virazel, and K. Bertels, "Emerging computing devices: Challenges and opportunities for test and reliability," in *2021 IEEE European Test Symposium (ETS)*, 2021, pp. 1–10.

[12] 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.

[13] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann, 2011. [Online]. Available: https://doi.org/10.1016/b978-0-12-369529-1.x5001-0

[14] 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.

[15] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 516–527. [Online]. Available: https://doi.org/10.1145/1250662.1250726

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

[17] 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.

[18] 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

[19] 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.

[20] 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.

[21] ——, "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.

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

[23] 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

[24] 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

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

[26] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos, "Differential fault injection on microarchitectural simulators," in *2015 IEEE International Symposium on Workload Characterization*, 2015, pp. 172–182.

[27] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527–1538, 2010.

[28] M. Ebrahimi, A. Evans, M. B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi, "Comprehensive analysis of sequential and combinational soft errors in an embedded processor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1586–1599, 2015.

[29] 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.

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

[31] ——, "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

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

[33] N. N. Sadler and D. J. Sorin, "Choosing an error protection scheme for a microprocessor's l1 data cache," in *2006 International Conference on Computer Design*, 2006, pp. 499–505.

[34] I. Alam, C. Schoeny, L. Dolecek, and P. Gupta, "Parity++: Lightweight error correction for last level caches," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 114–120.

[35] M. K. Qureshi and Z. Chishti, "Operating secded-based caches at ultra-low voltage with flair," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–11.

[36] Y. Ko, R. Jeyapaul, Y. Kim, K. Lee, and A. Shrivastava, "Guidelines to design parity protected write-back l1 data cache," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[37] M. Manoochehri and M. Dubois, "Accurate model for application failure due to transient faults in caches," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2397–2410, 2016.

[38] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. R. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 293–305.

[39] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and avf estimation revisited," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2010, pp. 477–486.