Priority Based Soft-Solution for Operating System Integrity Due To Randomized Hardware Faults At The Transistor Level

Alon Hillel-Tuch, Aspen Olmsted
Department of Computer Science
New York University Tandon School of Engineering
Brooklyn, NY 11201
ah5647@nyu.edu, aspeno@nyu.edu

***Abstract*— Hardware failures occur more often than we think. Environmental hazards such as electrical fires and liquid spills are easily detected and measured. Programming errors and defective hardware components (such as hard disk spindle defects) often lead to invalid operations, and we understand how and why. However, other less predictable forms of environmental stress such as radiation, thermal influence, and energy fluctuations exist and can induce hardware faults. We will explore what happens when a soft error, such as a bit-flip modified Word, remains valid to the Operating System. We will propose a priority-based scheduling solution to detect these faults at the software level with minimal overhead.**

Keywords— data integrity, error detection, bit flipping, hardware faults, system security, self-healing operating systems.

I. INTRODUCTION

Memory faults can manifest when there is no detectable hardware defect, be temporal, and not predictably replicable. Recovering from these errors at the Operating System level is non-trivial. However, once personal computing machine ownership became ubiquitous, researchers developed hardware error correction solutions that remain foundational in mainstream hardware-based Error Correction Controllers (ECC) memory applications[1].

ECC's are an elegant hardware solution to detect memory faults. Consumer-grade memory up until and including DDR4 predominantly did not contain ECC. The JEDEC standard for DDR5 has made an ECC variant part of the standard [2]. However, this would be an on-die solution that detects on-die errors, such as bit-flipping, and does not protect data in transit between the on-die memory controller and the central processing unit and is not as comprehensive as a DIMM-wide ECC solution. Consumer hardware does not have advanced ECC and relies on software-based solutions, if at all. Research looking to identify the balance between the overhead costs from software-based error detection, and expected gains from data integrity, has implicitly highlighted that the balance is subjective to the computational task [3].

The organization of the paper is as follows. In section II, we discuss our motivating example. Section III describes the related work and the limitations of current methods. Section IV describes how we developed our simulation. Section V drills into the data gathered in our experimentation and provides data visualizations. Section VI provides further analysis. Finally, we conclude in Section VII and discuss future work.

II. MOTIVATING EXAMPLE

We postulate that memory experiencing randomized hardware faults with no detection will remain prevalent in the foreseeable future due to legacy hardware and detection solutions in the consumer space. Research has explored self-healing operating systems using a micro-kernel structure [4]. Unfortunately, even with modern operating systems moving away from pure monolithic structures, they do not possess all the necessary components for implementing a strict micro-kernel operating system with fault-tolerance such as Minix3 [5].

Furthermore, specific faults can result in data that remains syntactically valid. Without a fault condition signaled on the CPU bus, there is no interrupt and no operating system intervention, potentially resulting in unexpected behavior. This behavior can compromise system security, popularized by case studies like Dinaburg's bitsquatting research [6].

Current soft solutions, while compelling, are complex and do not differentiate between critical and non-critical data, causing increased computational complexity due to broad-based redundant multithreading and space complexity [7-9]. As such, we propose that a soft-solution using a simple priority-flag-based scheduler will provide a more balanced solution for randomized hardware faults. Our solution will not only reduce operating system hard-faults but also enhance system security from attacks such as RowHammer [10] and other disturbance-based error attacks [11] [12].

III. RELATED RESEARCH

Research into software-based memory error detection emphasizes creating safe access solutions, such as MEDS, versus addressing hardware faults [13]. Hiser et al. focused on the programming language and the prevention of memory overwrites (a common form of memory error in non-memory safe languages), providing a solution and intermediary protection when only compiled binaries exist. However, MEDS does not protect seemingly valid data, which can occur in a bit-flip, nor does it genuinely acknowledge the hardware level, mainly remaining in the virtualized memory address space. Our research shares common challenges the MEDS team faced in handling low-overhead translation of their monitor.

Savaria and Velazco developed a comprehensive suite of error detection mechanisms that can complement and, in

some cases, replace hardware solutions [14]. Their answer towards complete bit-flip detection is highly applicable to industrial and memory-critical deployments. The computational and space cost of an instruction-duplication-based or signature-based technique is generally acceptable in these data-critical situations. However, in consumer environments, the overhead may not be considered tolerable when contrasted to the relatively high degree of non-critical data used in computations and expectation of low latency (responsiveness). The solution resulted in a "speed decrease by a threefold factor and required memory is four times larger" (p. 3517).

Additional work from independent researchers and cybersecurity firms around DinasBurg's bitsquatting concept has demonstrated that consumer-grade hardware is particularly susceptible due to missing error detection and the repetitive nature of specific tasks [15, 16]. Hax identified a core example, the Networking Time Protocol (NTP) servers. Windows synchronizes to the NTP servers regularly, creating a substantial sample size such that the expected value is still significant even with a low probability of error. For example, the 2038 problem is exploitable, causing an overflow in the signed 32-bit integer used to store time [17]. Research aptly highlights the vulnerabilities in systems but has not provided a realistically implementable soft solution.

IV. HYPOTHESIS AND PARAMETERS

We will conduct a simulation that artificially injects hardware faults and will measure correction, space complexity, and time complexity to demonstrate that an operating system-level scheduler for critical data is not only feasible but highly advised. Table I depicts computed Berger code check-bits for high priority words using a modified Berger code system.

TABLE I. BIT 0 ERROR (SINGLE)

Original Word	Bit Zero Error Word	Bit One Error Word
101 10	**0**01 10	1**1**1 10
101 10	10**0** 10	101 1**1**
101 10	101 0**0**	

We created a simulation of unique 8-bit memory operations (n = 4,729,000), a randomized single-bit soft-error rate, and a randomized priority classification. Our analysis will compare error correction against the increase in computational overhead between the historical method of operation and our scheduler-based parity-bit correction mechanism with the scheduler. We classify 15% of all operations as high priority (critical). We will use the measured relationship to extrapolate an appropriate priority classification for the scheduler. *No Detection* means no active mechanism, *Enhanced Detection* means we only apply our strategy on priority classified operations, and *Full Detection* means we use our approach on all memory operations.

We store the parity of a Word as a single bit, and upon retrieving the memory, we recalculate parity and compare it to the value stored in the bit. The priority-bit is OS-directed, and the expectation is that programs will declare which operations are priority-based, which the scheduler will then assign services.

Intel initially discovered that alpha-particle bombardment is a measurable cause of soft errors in DRAM [18]. IBM's research further quantified the observation into a probabilistic model that considered factors beyond just alpha-particles, dramatically increasing the probability of a soft error [19]. 1GB SEC ECC Memory will fail at a rate of 900 fails per 10,000 systems per three years, estimated at one mistake per month for every 256MB of ram. Using this information, we assign a rate for total single-bit errors introduced artificially in the simulation ($Error\ Probability\ Factor = 1.6 \times 10^{-5}, expected\ max\ error = 7.5 \pm 1.5$). These factors generate a random probability that approaches the qualifiers. A single parity bit is stored for each Word right now as part of an independent C Struct, but a production implementation would be native to the operating system.

V. EMPIRICAL EVIDENCE

Figure I illustrates the overhead variation between the three different strategies. To calculate parity, we must traverse the full n-length of the Word Θ(n). The additional overhead of Enhanced Detection follows the equation $S + (P \times S)$ *w*ith P equal to the probability of a priority designation and S equal to the number of original operations plus two (one write operation and one read operation of priority bit). Complete detection equals twice the number of initial steps (a constant set as the number of bits divided by four) multiplied by two, plus two. Computational Complexity is Θ(|Word|)**.** Space complexity is Θ(1) due to storage of a single parity bit and a single priority bit regardless of the size of Word.

FIGURE I. COMPUTATIONAL OVERHEAD ANALYSIS

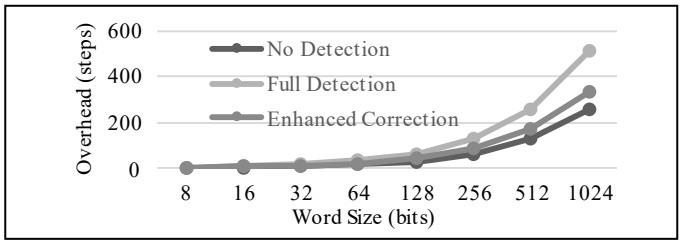
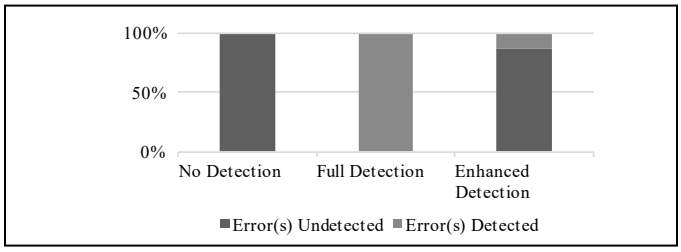


Figure II demonstrates that we detected only one error out of the eight introduced, an 87.5% miss rate. Again, a logical value, as it should follow P=.15 quite closely.

FIGURE II. ERROR DETECTION ANALYSIS



VI. DISCUSSION AND ANALYSIS

The miss rate is high; however, the seriousness of the error is substantially lower as it is a non-priority miss. The additional overhead involved in the enhanced operation follows $(P \times S)$, which is directly related to the size of the Word and the percentage of Priority operations. By differentiating between priority and non-priority, we can selectively choose when to incur additional time and space complexity to provide data integrity. We believe this to be particularly beneficial to non-

urgent background processes that use system downtime to perform tasks (such as time synchronization).

VII. CONCLUSION AND FUTURE WORK

Implementing a scheduling system to handle flagged data validity checks appears to have merit. It is warranted to explore existing widespread Operating Systems implementation as it would require kernel-level modifications and a discussion on good-actor behavior by developers, both challenging potential feasibility.

REFERENCES

[1] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM Journal of Research and Development,* vol. 28, no. 2, pp. 124-134, 1984-03-01 1984, doi: 10.1147/rd.282.0124.

[2] *DDR5 SDRAM*, JEDEC, Jul 2020 2020. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5 (Access Date: June 19th 2021).

[3] U. Schiffel, A. Schmitt, M. Süßkraut, and C. Fetzer, "Software-Implemented Hardware Error Detection: Costs and Gains," in *2010 Third International Conference on Dependability*, 2010-07-01 2010: IEEE, doi: 10.1109/depend.2010.16.

[4] F. M. David and R. H. Campbell, "Building a Self-Healing Operating System," in *Third IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC 2007)*, 2007-09-01 2007: IEEE, doi: 10.1109/dasc.2007.22.

[5] A. S. Tannenbaum. "Minix 3." Open-Source. http://www.minix3.org/ (Access Date: July 7th 2021).

[6] A. Dinaburg, "Bitsquatting - DNS Hijacking without Exploitation," in *BlackHat*, Las Vegas, Nevada, 2011: Raytheon Company. [Online]. Available: http://media.blackhat.com/bh-us-11/Dinaburg/BH_US_11_Dinaburg_Bitsquatting_WP.pdf (Access Date: June 17th 2021).

[7] B. Döbel, H. Härtig, and M. Engel, "Operating system support for redundant multithreading," in *Proceedings of the tenth ACM international conference on Embedded software - EMSOFT '12*, 2012-01-01 2012: ACM Press, doi: 10.1145/2380356.2380375.

[8] C. Borchert, H. Schirmeier, and O. Spinczyk, "Generative software-based memory error detection and correction for operating system data structures," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013-06-01 2013: IEEE, doi: 10.1109/dsn.2013.6575308.

[9] C. Borchert, H. Schirmeier, and O. Spinczyk, "Generic Soft-Error Detection and Correction for Concurrent Data Structures," *IEEE Transactions on Dependable and Secure Computing,* vol. 14, no. 1, pp. 22-36, 2017-01-01 2017, doi: 10.1109/tdsc.2015.2427832.

[10] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 39, no. 8, pp. 1555-1571, 2020-08-01 2020, doi: 10.1109/tcad.2019.2915318.

[11] S. Govindavajhala and A. W. Appel, "Using memory errors to attack a virtual machine," in *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*: IEEE Comput. Soc, doi: 10.1109/secpri.2003.1199334.

[12] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014-06-01 2014: IEEE, doi: 10.1109/isca.2014.6853210.

[13] J. D. Hiser, C. L. Coleman, M. Co, and J. W. Davidson, "MEDS: The Memory Error Detection System," Berlin, Heidelberg, 2009: Springer Berlin Heidelberg, in Engineering Secure Software and Systems, pp. 164-179.

[14] B. Nicolescu, Y. Savaria, and R. Velazco, "Software detection mechanisms providing full coverage against single bit-flip faults," *IEEE Transactions on Nuclear Science,* vol. 51, no. 6, pp. 3510-3518, 2004-12-01 2004, doi: 10.1109/tns.2004.839110.

[15] R. R. Salazar Oscar, "Ghost in the browser: broad-scale espionage with bitsquatting," in *Kaspersky Security Analyst Summit*, Singapore, B. Fox, Ed., 2019: Kaspersky. [Online]. Available: https://resources.bishopfox.com/files/slides/2019/Kaspersky%20SAS-Ghost-in-the-Browser-Broad-Scale-Espionage-with-Bitsquatting-10Apr2019-%20Slides.pdf (Access Date: June 22nd 2021).

[16] R. Hax, "Bitsquatting Windows.com," vol. 2021, ed, 2021. Available: https://remyhax.xyz/posts/bitsquatting-windows/ (Access Date: July 21st 2021).

[17] D. Spinellis, *Code Quality: The Open Source Perspective*. Boston, MA: Addison-Wesley Educational, 2006.

[18] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices,* vol. 26, no. 1, pp. 2-9, 1979-01-01 1979, doi: 10.1109/t-ed.1979.19370.

[19] T. J. Dell, "A White paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," 11/19, 1997.